UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Funciones de activación ruidosas en redes neuronales recurrentes.

Pablo Saucedo de Miguel
Tutor: Luis Fernando Lago Fernández

JULIO 2020

Funciones de activación ruidosas en redes neuronales recurrentes.

AUTOR: Pablo Saucedo de Miguel
TUTOR: Luis Fernando Lago Fernández

Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio de 2020

Resumen

Este Trabajo de Fin de Grado pretende hacer un análisis sistemático de la influencia del ruido aplicado en funciones de activación de redes neuronales recurrentes, concretamente en las *Elmann RNN*. El ruido es una estrategia de regularización, es decir, penaliza la complejidad de los modelos para reducir el *overfitting* y así lograr mejorar la capacidad de generalización. La forma en la que simplifica los modelos es generando inestabilidad en las neuronas que se sitúen en la región de máxima pendiente de la función de activación, en este caso la tangente hiperbólica, para así provocar que las neuronas se desplacen a las regiones donde el ruido tiene menos influencia: las regiones con derivada 0.

Debido a su condición de regularización, su uso únicamente es útil para problemas que sufren de sobre ajuste, en los que genera que las neuronas se polaricen adquiriendo valores de 1 o -1. Este efecto provoca que, por un lado, se mejore la pérdida para el conjunto de validación y, por otro lado, que aumente la interpretabilidad de la red, utilizando como medida la entropía.

Otros regularizadores, como el L1 o el L2, penalizan los pesos de las neuronas repartiendo la penalización de complejidad entre ellos. Su efecto es similar al de *feature selection*, seleccionando las neuronas necesarias para resolver el problema y apagando el resto. La coexistencia de estos regularizadores, junto con el ruido, podría lograr aumentar aún más la interpretabilidad al apagar neuronas que se sitúen en valores de máxima pendiente de la función de activación. Este trabajo demuestra que ambas técnicas no consiguen coexistir, a pesar de introducir estrategias de ruido específicas para intentarlo.

Se exploran distintas estrategias para introducir ruido, con el objetivo de optimizar tanto la pérdida como la interpretabilidad, resultando el ruido dependiente del estado actual como la estrategia óptima dentro de este trabajo. Por último, se aplican los modelos recurrentes a un problema de generación de texto.

Abstract

This Bachelor Thesis aims to make a systematic analysis of the influence of noise applied on activation functions of recurrent neural networks, specifically *Elmann RNN*. Noise is a regularization strategy, that is, it penalizes the complexity of the models to reduce ovefitting and thus improve the generalization capacity. The way in which it simplifies the models is by generating instability in the neurons that are located in the región of maximum slope of the activation function, in this case, the hyperbolic tangent, in order to cause the neurons to move to the regions where the noise has less influence: the 0 derivative regions.

Due to its regularization condition, its use is only useful for problems that suffer from overfitting, in which it causes neurons to polarize, acquiring values of 1 or -1. This effect causes, on one hand, the loss for the validation set to be improved and, on the other hand, to increase the interpretability of the network, using entropy as a measure.

Other regulators, such as L1 or L2, penalize the weights of neurons by distributing the complexity penalty among them. Its effect is similar to that of feature selection, selecting the neurons necessary to solve the problem and turning off the rest. The coexistence of these regulators, together with the noise, could further increase the interpretability by turning off neurons that are at maximum slope values of the activation function. This work shows that both techniques cannot coexist, despite introducing specific noise strategies.

Different strategies for introducing noise are explored, with the aim of optimizing both the loss and the interpretability, resulting in noise that is dependent on the current state as the optimal strategy in this work. Finally, recurrent models are applied to a text generation problem.

Palabras clave

Aprendizaje profundo, redes neuronales recurrentes, función de activación, ruido, sobre ajuste, regularización, complejidad.

Keywords

Deep Learning, recurrent neural networks, activation function, noise, overfitting, regularization, complexity.

Agradecimientos

Quiero agradecer a mi tutor, Luis Lago Fernández por toda su dedicación, interés y confianza que ha depositado en mi trabajo. Además, junto con él, quiero agradecer a Christian Oliva toda la ayuda y todos los ratos de debate interesante que me ha aportado. He aprendido gracias a ellos la belleza que supone realizar una investigación y lo útil que resultan las sesiones de análisis conjuntas de resultados, que me han ayudado a ampliar mucho más mis conocimientos en el campo de las redes neuronales.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

INDICE DE ECUACIONES

1 Introducción

1.1 Motivación

En este trabajo de fin de grado voy a analizar el efecto del ruido en redes neuronales recurrentes. Las redes neuronales son un algoritmo englobado en el mundo de la Inteligencia Artificial, más concretamente en el conjunto de técnicas de aprendizaje automático, conocido como *Machine Learning*. Están inspiradas en la estructura biológica del cerebro humano, donde la neurona es la unidad básica de cómputo, y pretenden imitar su proceso de aprendizaje.

Las redes neuronales son un algoritmo realmente potente debido a su condición de aproximador universal [1], capaces de modelar abstracciones de los datos utilizando transformaciones continuas y no lineales. Su estructura está compuesta por neuronas dispuestas en capas secuenciales interconectadas por enlaces que se conocen como pesos, que permiten el flujo de los datos.

Particularmente, las redes neuronales recurrentes son una especialización de las redes convencionales, que permiten tratar los datos de forma secuencial, ya que cuentan con un estado interno que actúa como una memoria. Por tanto, los datos de entrada serán tratados de forma dependiente, es decir, para dar una salida se tendrá en cuenta tanto la entrada como el estado interno de la red.

Uno de los problemas más comunes en las redes neuronales es el sobre ajuste, *overfitting*. Ocurre cuando el modelo consigue recordar los datos de entrenamiento, consiguiendo buenos resultados en ese conjunto, pero en consecuencia pierde capacidad para generalizar con datos de entrada desconocidos. Para enfrentar este problema existen diferentes estrategias, entre las que destacan las técnicas de regularización, donde se incluye el ruido.

La condición de regularización del ruido va a tener un impacto en la capacidad de generalización de la red y, además, su aplicación va a mejorar su interpretabilidad [2]. Como consecuencia directa, la red puede ser interpretada como un autómata finito [3]. Por otro lado, el ruido también ha sido utilizado para la labor opuesta, es decir, para sacar a las neuronas de las zonas no derivables con el objetivo de optimizar el aprendizaje [4].

El ruido consiste en introducir perturbaciones, moduladas por una función de probabilidad, en algún punto de la red. Esta técnica ha sido principalmente utilizada como regularizador, cuyo objetivo es penalizar la complejidad de un modelo para favorecer la labor de generalización. Su efecto es el de añadir inestabilidad al modelo, forzando a las neuronas a adoptar los valores más estables para poder realizar una labor de entrenamiento óptima. En mi caso, el ruido será introducido en la función de activación, que será la tangente hiperbólica, cuyos valores más estables son sus límites superior e inferior, es decir, 1 o -1.

1.2 Objetivos

En este trabajo propongo realizar un estudio sistemático del ruido introducido en funciones de activación, más concretamente en redes neuronales recurrentes. Identificaré las ventajas que aporta el ruido, aplicándolo de diversas formas ya que, dependiendo de la función de probabilidad utilizada, su amplitud, el momento de aplicación y su dependencia con otros componentes, se pueden obtener diferentes resultados. Los objetos de estudio serán los siguientes:

1. Estudio del efecto del ruido en la capacidad de generalización de la red. Como consecuencia del uso del ruido como regularizador para reducir el *overfitting*.
2. Estudio del efecto del ruido en la interpretabilidad de la red. Como consecuencia de desplazar los valores de activación de las neuronas a los valores extremos de la función de activación.
3. Aplicar el ruido en la resolución de un problema de generación de texto. Uno de los casos de uso más comunes de las redes neuronales recurrentes. [5]

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- Estado del Arte dónde voy a exponer claramente lo que son las redes neuronales, centrándome en las redes neuronales recurrentes y su aplicación en el problema de generación de texto. Además, profundizaré más en la aplicación del ruido.
- Diseño del modelo, donde se explicará la arquitectura de la red a utilizar y el ruido que será introducido. Se fijarán las condiciones generales de todos los experimentos que van a ser realizados.
- Desarrollo de los experimentos propuestos. Se hará de manera secuencial para comprender el flujo de investigación, exponiendo los problemas encontrados y las soluciones utilizadas. Se realizará un barrido desde las pruebas más generales a las más específicas.
- Resultados, donde se expondrán los resultados de las distintas pruebas propuestas.
- Conclusiones, donde se especificarán las ventajas que supone introducir ruido y donde se planteará el trabajo futuro.

2 Estado del arte

2.1 Redes Neuronales

Las redes neuronales son un modelo computacional que busca imitar el comportamiento neuronal del cerebro humano. Su origen se remonta al año 1943, en el que el neurofísico Warren McCulloch y el lógico Walter Pitts propusieron el primer modelo basado en circuitos eléctricos [6]. Tras años de investigación en el que se involucraron multitud de equipos, finalmente en 1959 se desarrollaron los primeros modelos computacionales aplicables a problemas reales: el perceptrón [7], el adaline y el madaline [8],

Desde entonces el desarrollo ha sido imparable en este campo, sobre todo gracias a la evolución del *Deep Learning*, que engloba las redes neuronales profundas, es decir, arquitecturas con muchas capas intermedias. Este tipo de redes ha permitido ampliar aún más las aplicaciones de las redes neuronales, ya que son capaces de tratar datos más complejos con muy buenos resultados. Además, la investigación para el desarrollo de nuevas arquitecturas, técnicas de optimización o de regularización permite seguir aumentando las posibilidades que las redes neuronales ofrecen. En la Fig. 1 se sitúa el *Deep Learning* dentro del paradigma de la Inteligencia Artificial (AI).

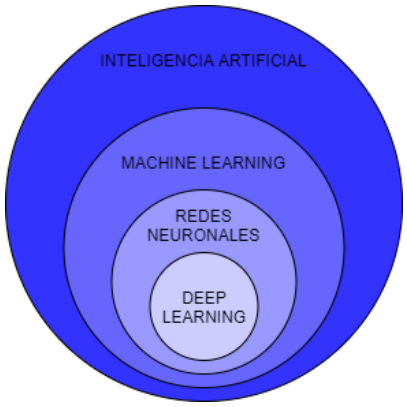


Figura 1: Mundo AI

Las redes neuronales están compuestas por una sucesión de capas de neuronas artificiales que se conectan unas a otras mediante unas conexiones denominadas pesos. En la optimización de estos pesos es donde se produce el aprendizaje porque, recordemos que al ser una técnica de *Machine Learning*, estos algoritmos tienen la capacidad de aprender sin necesidad de una programación explícita. Cada red neuronal tiene asociada una función de pérdida que tiene como objetivo evaluar la precisión de la red, es decir, conocer cuanto difiere la salida de la red con la salida real dada una entrada determinada. El aprendizaje se logra encontrando los pesos que minimizan la función de pérdida.

Este procedimiento se divide en tres partes bien diferenciadas:

1. Propagación hacia adelante: Las neuronas de la capa de entrada reciben los datos del caso de entrenamiento. A continuación, la información se propaga al resto de capas multiplicando la activación de la neurona de la capa de origen por el peso

correspondiente al enlace entre la neurona de origen y la de destino. Cuando las neuronas de la capa de destino reciben todos los impulsos, hay que aplicar la función de activación para obtener el valor final de activación de la neurona. La propagación hacia adelante finaliza cuando las neuronas de la capa de salida tienen su correspondiente activación:

$$y' = WX + b$$
$$y = f(y')$$

Ecuación 1: Cálculo de la activación de las neuronas

Siendo *X* y *W* los valores de activación y los pesos de la capa de origen, respectivamente; *b* corresponde al sesgo; *f(y´)*, la aplicación de una función de activación determinada sobre la salida de la fórmula anterior, e *y* los valores finales de activación de la capa de destino.

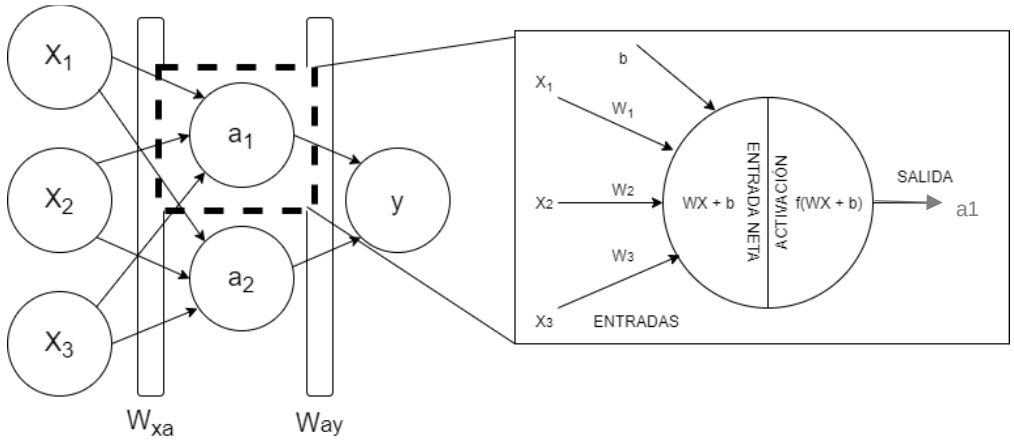


Figura 2: Funcionamiento de una red neuronal

En la Fig.2 se muestra la estructura básica de una red neuronal y el proceso de cálculo del valor de activación de una neurona.

2. Propagación hacia atrás. En este paso es donde se logra el aprendizaje. Esta técnica, conocida como *backpropagation* [9], se desarrolló en 1986 y conforma el algoritmo estándar. Se inicia calculando el error entre la salida calculada por la red y la salida esperada, para continuar aplicando un determinado optimizador para realizar la tarea de descenso por gradiente, *gradient descent*, de la función de coste para aproximarse a su mínimo, ilustrado en la Fig. 3. Este error se propaga hacia atrás calculando los gradientes del error con respecto a los parámetros de la red.

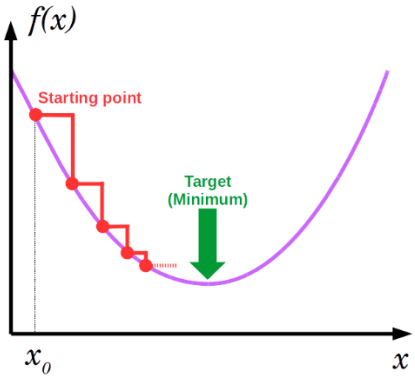


Figura 3: Representación del descenso por gradiente
Fuente: *FreePng*

3. Actualización de pesos. En este último paso, se utiliza el gradiente del error respecto a cada uno de los parámetros para recalcular los pesos, así como los sesgos. Esta actualización se puede llevar a cabo de distintas formas e influyen hiperparámetros que se explicarán más adelante.

2.2 Redes Neuronales Recurrentes

Las redes neuronales son generalmente utilizadas como un algoritmo de aprendizaje supervisado, es decir, son entrenadas con un par de valores entrada-salida conocidos. Estos pares de valores se tratan de manera independiente, haciendo que la red actúe como un algoritmo determinista una vez que está entrenada. Por el contrario, las redes neuronales recurrentes RNN tienen en cuenta las entradas anteriores para dar una salida, es decir, tratan los datos de manera dependiente. Una vez entrenada, y dada una entrada, la salida variará dependiendo de las entradas anteriores [10].

Estas redes son muy potentes debido a un estado interno que actúa como una memoria, lo que permite la labor de recordar y procesar información pasada. Aun así, las RNN comunes, las *Vanilla RNN*, no son una herramienta muy utilizada en el mundo AI debido a la dificultad que supone entrenarlas por la inestable relación que existe entre los parámetros de entrada y el estado interno, manifestado en un problema común en estas redes: *vanisihing/exploding gradients* [4]. Este problema es muy habitual en redes neuronales profundas, debido a que, entre las distintas capas, el gradiente se transmite multiplicándose. Si el valor de este gradiente es mayor que 1, el valor final tenderá a infinito. Ocurre lo contrario cuando el gradiente es menor que 1, ya que el valor tenderá a 0.

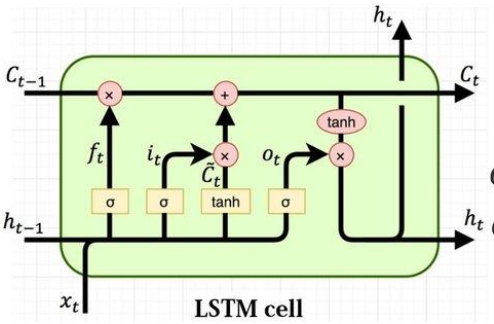
Las redes neuronales recurrentes tienen una tendencia a sobreajustar [11]. Este problema, conocido habitualmente como *overfitting*, ocurre cuando la red es capaz de recordar los patrones de entrada con su salida correspondiente, perfeccionando la labor de entrenamiento a costa de penalizar la capacidad de generalización, es decir, la capacidad de dar una salida correcta dada una entrada desconocida. Este problema puede ser consecuencia de diversos factores, entre los que destaco tres [12]:

1. Complejidad alta del modelo. Reflejada en el número de parámetros a optimizar, que aumentarán al introducir más neuronas o capas a la red.
2. Insuficientes datos de entrenamiento. Será más sencillo para la red recordar pocos patrones.

3. Muchas épocas de entrenamiento. El aprendizaje de una red es siempre creciente, por lo que se irá continuamente aprendiendo los patrones. Si el entrenamiento es lo suficientemente largo y la red lo suficientemente compleja, todos los patrones serán recordados.

Basándose en la arquitectura recurrente de las RNN, existen otros tipos de redes con una precisión mayor y que solventan el problema del *vanishing/exploding gradient*, como las LSTMs o las GRUs.

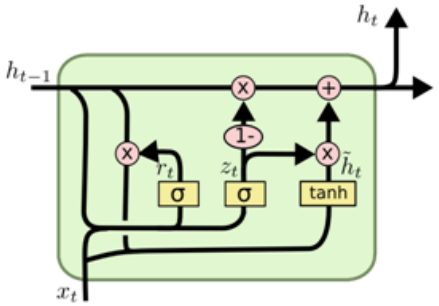
Las LSTM (*Long Short Term Memory)*, representadas en la Fig.4, son otro tipo de red neuronal recurrente [13]. La idea es similar a la de una RNN convencional, ya que cuentan con un estado interno, conocido como *cell*, que actúa como una memoria que almacena las relaciones entre los elementos de la entrada. La diferencia principal con respecto a una RNN estándar reside en que este estado está administrado por tres reguladores, conocidos como puertas (*gates)*: la *input gate*, que controla en qué medida es introducido un dato en el *cell*; la *forget gate*, que controla el tiempo que se mantiene un dato en el *cell*; y la *output gate*, que determina la influencia de los datos en la salida final.



$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$
$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$
$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$

Figura 4: Estructura y ecuaciones de una LSTM
Fuente: ***researchgate.net***

Una particularización de las LSTM son las conocidas como GRUs (*Gated Recurrent Units)* [14] ilustrada en la Fig.4. Este tipo de red tiene menos parámetros que las LSTMs y carecen de la *output gate*, lo que las hace un modelo más sencillo y con una precisión similar para ciertas tareas como procesamiento de lenguaje natural [15]. A pesar de esto, las LSTM son una arquitectura más potente [16] [17].



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figura 5: Estructura y ecuaciones de una GRU
Fuente: ***mc.ai***

En este trabajo, aunque estos últimos tipos de arquitectura logren mejores resultados para el problema planteado en el apartado 3.6, haré uso de las redes recurrentes estándar. Esta decisión se debe a que son un modelo más sencillo que me va a permitir tener más control sobre su estado interno y sobre la influencia del ruido.

2.3 La generación de texto

La generación de texto es una de las aplicaciones más comunes de las redes neuronales recurrentes [5], pudiendo trabajar a nivel de palabra o a nivel de carácter, como en este caso. Este problema consiste en entrenar un modelo que reciba un carácter como entrada y devuelva como salida el carácter con mayor probabilidad de ir a continuación. Por tanto, es un problema con una salida completamente probabilística.

Dada una cadena s = $c_1, \dots, c_T$ compuesta por T caracteres de un vocabulario W, la probabilidad de salida de la cadena es [18]:

$$P(s) = \prod_{t=1}^{T} P(c_t |\ c_1, \dots, c_{t-1})$$

Ecuación 2: Probabilidad de un carácter en la generación de texto

Para realizar operaciones con caracteres es fundamental realizar el proceso de codificación. Existen muchas técnicas para ello pero, una de las más habituales y es la elegida para este estudio, es la *One Hot Encoder*. Esta codificación transforma cada carácter en un vector disperso de tamaño la longitud de W con un único valor a 1 situado en la posición de dicho carácter. Cuanto mayor sea el vocabulario W, más grandes serán estos vectores y más complicado será el problema.

2.4 Uso del ruido en redes neuronales

Toda la investigación de este trabajo gira en torno al ruido introducido en funciones de activación. Es necesario conocer qué es realmente y en qué puede beneficiar a un modelo. Es un concepto realmente sencillo de entender y de implementar, de ahí que sea interesante investigar sobre ello.

El ruido, para este trabajo, se utiliza para perturbar la función de activación de la red con un valor pseudoaleatorio definido por una función de probabilidad con una amplitud determinada. Una vez generado ese valor, existen multitud de formas de introducirlo dentro de las ecuaciones de propagación hacia adelante ya que, dependiendo del momento en el que se introduzca y/o la dependencia que tenga con otras componentes de la red, el resultado puede variar.

El ruido es originalmente un regularizador. Como todos los regularizadores lo que buscan es penalizar la complejidad de un modelo, pero difieren en la definición de "complejidad": los regularizadores clásicos, como L1 o L2, reducen los pesos de las conexiones para conseguir reducir la importancia de las neuronas, llevándolas a valores cero de activación o cercanos; otras técnicas como *drop-out*, buscan simplificar el modelo apagando ciertas neuronas en cada época de entrenamiento para complicar la tarea de memorización.

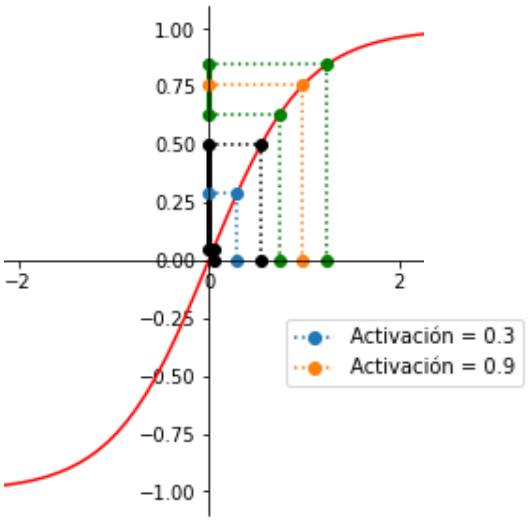


Figura 6: Inestabilidad generada por el ruido

Tal y como se observa en la Fig. 6, donde se muestran dos valores de activación diferentes a los que se aplica la misma cantidad de ruido, este afecta en mayor medida a las neuronas que se encuentran en la región de máxima pendiente de la función de activación. A medida que las neuronas se acerquen a los valores de polarización 1 o -1, el ruido tendrá menos impacto y la red ganará estabilidad, favoreciendo el aprendizaje. Por tanto, la forma que tiene el ruido de reducir la complejidad del modelo, es decir, simplificarlo, es llevándose las neuronas a las regiones de derivada próxima a cero para que adquieran valores cercanos a 1 o -1. Esta polarización va a ayudar a mejorar la interpretabilidad de la red [19]. Una vez que se logre ese efecto, se puede asemejar su comportamiento con el de una máquina de estados [3].

El otro uso que tiene el ruido es justamente el inverso. Debido a que el algoritmo de *backpropagation* requiere de la derivada de la función de activación para optimizar los parámetros, si las activaciones de las neuronas se sitúan en las regiones de los extremos, donde la derivada es muy cercana a 0, la actualización será mínima, impidiendo que se sigan optimizando los parámetros. El ruido en este caso se aplica para sacar a las neuronas de los valores 1 o -1, devolviéndolas a la región de máxima pendiente que permitirá actualizar los parámetros de la red para continuar optimizando el aprendizaje.

2.5 Desarrollo de redes neuronales

Las redes neuronales son un algoritmo de aprendizaje automático computacionalmente muy costoso, por lo que necesitan una capacidad de cómputo suficiente para poder desarrollar el proceso de entrenamiento. Esta propiedad es responsabilidad del procesador del equipo donde se ejecute.

Mayoritariamente los equipos cuentan con CPU *(Central Processing Unit)* que internamente cuenta con 2,4 u 8 ALUs y están destinados a un propósito general. Su operación básica de cómputo es entre escalares. Cuando se han requerido de mayor capacidad, se han desarrollado tanto las TPUs *(Tensor Processing Unit)*, con la operación entre vectores como operación básica; como las GPUs *(Graphics Processing Unit)* destinadas a las tareas más

pesadas, ya que cuentan con una arquitectura desarrollada para favorecer la paralelización y tienen las operaciones con matrices como operación básica de cómputo, lo que favorece claramente el entrenamiento de redes neuronales.

Como aún son escasos los equipos con GPU integradas, existen soluciones *Cloud* para lanzar los procesos en GPUs compartidas. Entre ellas destaca *Google Colab,* que ha sido la elegida para esta investigación, debido a que es gratuita, se integra directamente con *Google Drive* e incluye *Tensorflow/Keras* como *backend* de desarrollo nativo, al estar también desarrollado por *Google.*

Tensorflow[1] está disponible para los lenguajes Python y JavaScript, además de una versión Lite enfocada a su uso en sistemas embebidos, es actualmente el *framework* más utilizado. Además, en el mercado existen otras opciones para el desarrollo de redes neuronales, tales como *PyTorch*[2], desarrollado por *Facebook* y que está en pleno auge; o *Caffe*[3], un software libre desarrollado para multitud de lenguajes y fácil gracias a su gran nivel de abstracción.

Empresas como *Amazon*[4] o *Microsoft*[5] están añadiendo servicios de desarrollo de redes neuronales en sus kits de *Cloud.*

[1] https://www.tensorflow.org/

[2] https://pytorch.org/

[3] https://caffe.berkeleyvision.org/

[4] https://aws.amazon.com/es/sagemaker/

[5] https://docs.microsoft.com/en-us/cognitive-toolkit/

3 Diseño

3.1 Red utilizada

Las *Elmann RNN* son el tipo de red recurrente más sencillo que existe [10]. Su propósito es recibir una secuencia de vectores de entrada $(x_1, x_2, \dots, x_T)$ con los que calcular una secuencia de estados internos $(h_1, h_2, \dots, h_T)$ y una secuencia de salidas $(o_1, o_2, \dots, o_T)$:

Para t = 1 hasta T:

$$h_t = \tanh(W_{hx}x_t + W_{hh}h_{t-1} + b_h)$$
$$o_t = \text{softmax}(W_{oh}h_t + b_o)$$

Ecuación 3: Cálculo de la activación de las neuronas para una red de *Elmann*

Donde W_{hx} es la matriz de pesos entre la entrada y la capa oculta, W_{hh} define los pesos de la conexión recurrente, W_{oh} es la matriz de pesos entre capa oculta y salida y los vectores b_h y b_o son los sesgos de la red. En tiempo t = 1, la expresión h_{t-1} es sustituida por un vector especial de inicialización, llamado h_{init}. Por defecto, se aplica la función tangente hiperbólica como función de activación. [5]. Como la salida de la red será en codificación *One hot*, cada componente tendrá un valor de 0 o 1, por lo que habrá que aplicar la función *softmax* de salida.

Este tipo de redes están compuestas únicamente por tres capas: la de entrada, la oculta y la salida. Todas estas capas son *fully-connected*, es decir, una neurona de una capa determinada tiene conexión con todas las neuronas de la capa siguiente. Para adaptar esta red al problema de generación de texto, que detallaré en el apartado 3.6, la capa de entrada y la de salida deben de tener el mismo número de neuronas que el número de caracteres distintos que se contemplen, es decir, 88. Las neuronas en la capa oculta irán variando a lo largo de las pruebas.

3.2 Ruido utilizado

El ruido que voy a introducir en las ecuaciones va a ser generado por una función de distribución normal, centrada en 0 y tendrá como amplitud la definida por el parámetro *noise_level*, ilustrado en la Fig. 7.

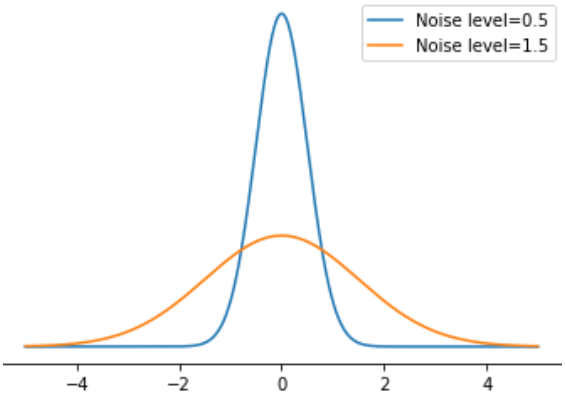


Figura 7: Generación de ruido con distribución normal

Siguiendo este criterio general, voy a introducir el ruido en diferentes momentos y voy a hacerlo dependiente de otros componentes de las ecuaciones. El ruido estará representado por el vector X_v en las ecuaciones y tendrá una longitud igual al número de neuronas en la capa oculta.

3.2.1 Ruido independiente

Esta estrategia es la más sencilla para introducir ruido. Consiste en introducirlo de forma independiente, afectando a todas las neuronas, y previamente al paso por la función de activación *tanh:*

Para t = 1 hasta T:

$$h_t = \tanh\left(W_{hx}x_t + W_{hh}h_{t-1} + \boldsymbol{X_v} + b_h\right)$$
$$o_t = \text{softmax}(W_{oh}h_t + b_o)$$

Ecuación 4: Ruido independiente

3.2.2 Ruido dependiente

En esta variante del primero, el ruido se introduce también antes de la función de activación, pero es dependiente del estado anterior en el que se encontrase la neurona, es decir, neuronas con una activación más cercana a cero, se verán menos afectadas que las neuronas cercanas a los valores de polarización:

Para t = 1 hasta T:

$$h_t = \tanh\left(W_{hx}x_t + W_{hh}h_{t-1} + \boldsymbol{X_v} \circ \boldsymbol{h_{t-1}} + b_h\right)$$
$$o_t = \text{softmax}(W_{oh}h_t + b_o)$$

Ecuación 5: Ruido dependiente

Donde ∘ representa el producto componente a componente entre ambos vectores.

3.2.3 Ruido posterior independiente

El siguiente tipo de ruido será introducido después de la función de activación, con lo que se conseguirá una mayor agresividad, lo que generará mayor inestabilidad.

Para t = 1 hasta T:

$$h_t = \tanh(W_{hx}x_t + W_{hh}h_{t-1} + b_h)$$
$$o_t = \text{softmax}(W_{oh}h_t + b_o) + \boldsymbol{X_v}$$

Ecuación 6: Ruido posterior independiente

3.2.4 Ruido dependiente del estado actual versión 1

Por último, las pruebas finales se harán con un ruido dependiente del estado actual de la neurona. El objetivo es similar al del caso dependiente del estado anterior: minimizar el impacto del ruido en neuronas con activaciones cercanas al 0. La diferencia reside ahora en ser más directos para tener en cuenta el estado de la neurona para la entrada proporcionada.

Para t = 1 hasta T:

$$h_t = \tanh((W_{hx}x_t + W_{hh}h_{t-1} + b_h) \circ (1 + \boldsymbol{X_v}))$$
$$o_t = \text{softmax}(W_{oh}h_t + b_o)$$

Ecuación 7: Ruido dependiente del estado actual v1

3.2.5 Ruido dependiente del estado actual versión 2

Esta forma de introducir ruido es similar a la estrategia anterior, pero soluciona un problema que tiene. Este consiste en que el estado actual aún no ha pasado por la función de activación y no está acotado, por lo que, si el estado es muy grande, el ruido también lo será. Lo que haremos entonces será calcular previamente la función de activación del estado actual para conseguir que se acote al rango [-1,1].

Para t = 1 hasta T:

$$h'_t = \tanh(W_{hx}x_t + W_{hh}h_{t-1} + b_h)$$
$$\boldsymbol{X'_v} = \boldsymbol{X_v} \circ \boldsymbol{h'_t}$$

$$h'_t = \tanh(W_{hx}x_t + \boldsymbol{X'_v} + W_{hh}h_{t-1} + b_h)$$
$$o_t = \text{softmax}(W_{oh}h_t + b_o)$$

Ecuación 8: Ruido dependiente del estado actual v2

3.3 Regularizadores

El ruido actúa como un regularizador para las neuronas, tal y como hemos mencionado anteriormente. Pero también puede resultar interesante utilizarlo junto con alguno de los regularizadores clásicos más utilizados: L1 y L2. Estas técnicas, además de contribuir a

evitar el *overfitting*, nos van a ayudar en la tarea de interpretabilidad como vamos a mostrar más adelante.

Como ya hemos mencionado anteriormente, la regularización lo que busca es penalizar la complejidad del modelo. Cada algoritmo entiende la complejidad de una manera diferente y plantea la solución para enfrentarla. Definamos los dos tipos de regularizadores con los que vamos a trabajar:

- Regularización L1: En este tipo de regularización, la complejidad se mide como la media en valor absoluto de los pesos del modelo. La regularización L1 lleva a cabo un proceso de *feature selection*, es decir, elimina, haciendo que los pesos se acerquen a cero, los atributos de la entrada que tienen poca influencia en la salida.

$$C = {}^{\lambda}/_{2M} * \sum_{i=1}^{N} |w_i|$$

Ecuación 9: Regularización L1

Siendo M el número de casos de entrenamiento y N el número de pesos.

- Regularización L2: En este otro tipo de regularización, lo que se penaliza es el cuadrado del valor de los pesos del modelo. Lo que busca es minimizar el efecto de la correlación entre atributos de entrada, llevando los pesos a valores muy pequeños. Esto fomentará la capacidad de generalización de la red.

$$C = {}^{\lambda}/_{2M} * \sum_{i=1}^{N} w_i^2$$

Ecuación 10: Regularización L2

Debido a su condición de regularizadores, ambos ayudarán en el objetivo de mejorar la capacidad de generalización de la red, Además, gracias a su forma de penalizar la complejidad del modelo, las neuronas reducirán su activación, incluso llegándose a apagar, consiguiendo mejorar la interpretabilidad de la red, otro de los objetivos de este trabajo. Particularmente para esta última tarea, L1 va a ser más útil ya que realiza un proceso de selección de neuronas, una estrategia más agresiva que la que adopta L2, que reparte la penalización entre muchas neuronas y provoca que todas se activen menos.

3.4 Optimizadores

Las redes neuronales necesitan un optimizador con el que poder entrenar las matrices de pesos y los sesgos. Estas funciones de optimización definen la estrategia para lograr minimizar dichos parámetros para encontrar el mínimo de la función de coste.

Los optimizadores que utilizaremos son los siguientes:

- *Stochastic Gradient Descent, SGD:* Se trata de un método iterativo de optimización que utiliza una adaptación estocástica del descenso por gradiente clásico, ya que utiliza un subconjunto aleatorio de los datos para realizar la actualización de los pesos. Esto genera una mejora considerable en el tiempo de entrenamiento, pero por otro lado, es más habitual que encuentre algún mínimo local [4]. Entre los muchos

hiperparámetros que utiliza, nos centraremos en *learning rate* y *clip-value*, que definiremos más adelante.

- *Adaptive moment estimation, Adam:* Este optimizador es una implementación de SGD que trata de realizar un aprendizaje para calcular los hiperparámetros más apropiados basándose en el primer y segundo momento de los gradientes, por tanto, es adaptativo. Adam proporciona una actualización de hiperparámetros más lenta pero más efectiva, favoreciendo la convergencia.

3.5 Hiperparámetros

Las redes neuronales cuentan con dos tipos de parámetros: los entrenados durante el aprendizaje, inicializados habitualmente a valores aleatorios y que se adaptan durante el proceso de optimización; y los hiperparámetros, que se han de definir al crear el modelo. Los optimizadores requieren de los hiperparámetros para realizar su labor, por lo que será necesario encontrar los valores que optimicen el entrenamiento. Adam no requiere de ningún trabajo previo ya que se encarga internamente de ajustarlos, pero para SGD hemos necesitado definir dos hiperparámetros que tienen gran influencia:

- Tasa de aprendizaje, *learning rate*: Este parámetro, existente en cualquier método de optimización, define la influencia de las correcciones de pesos y sesgos en la actualización de los mismos. De forma gráfica, define el tamaño del salto que se va a realizar en cada paso de optimización. Con una tasa de aprendizaje baja, nos aseguramos llegar al mínimo local de la función, pero serán necesarias muchas iteraciones. Por otro lado, una tasa de aprendizaje alta podrá causar divergencia en las actualizaciones.

- *Gradient Cliping, clip value:* Debido al algoritmo de *backpropagation*, los gradientes de los errores se propagan hacia atrás como producto. Por tanto, si el valor del gradiente es superior a 1, puede suceder que, tras muchas iteraciones, el gradiente crezca exponencialmente y se produzca el problema de *exploding gradient.* Al utilizar redes neuronales recurrentes que son redes profundas, es mucho más común este problema. El *clip value* permite subsanar este error limitando el crecimiento del gradiente.

Para encontrar los valores óptimos para el problema (apartado 3.6) de los hiperparámetros mencionados, he realizado una búsqueda en rejilla para conocer con qué valores de *learning rate* y de *clip value* obtenemos los mejores resultados en pérdida de validación. La tabla de resultados se puede encontrar en el Anexo A y va a definir los hiperparámetros de las pruebas del apartado 4.

Existen más hiperparámetros que permiten modular el comportamiento de las redes. Algunos ejemplos pueden ser el número de neuronas por capa o el número de épocas de entrenamiento, que las definiré a lo largo de las pruebas; el número de capas, que al utilizar una *Elmann RNN* (apartado 3.1) este valor está fijo a 3; la función de activación a utilizar, en mi caso será la *tanh* o el tamaño de *batch*, que fijaré a 32.

3.6 Descripción del problema

El problema que voy a utilizar para probar la influencia del ruido es una aplicación muy común de las redes neuronales: la generación de texto. En mi caso, voy a utilizar una de las novelas más importantes de nuestra literatura: *El Ingenioso Hidalgo Don Quijote de la Mancha* para entrenar las redes. Descargué la novela en texto plano aquí[6]

La novela tiene en total 1038397 caracteres. Existen 88 caracteres diferentes, en los que se incluyen todas las letras del castellano en mayúscula y minúscula, signos de puntuación, espacios, saltos de línea y demás símbolos. No he llevado a cabo ninguna tarea de limpieza de datos para buscar unos resultados lo más naturales posibles. Los caracteres más inusuales, al tratarse de un problema probabilístico, tendrán menos opciones de aparecer en las predicciones.

Las redes neuronales únicamente aceptan entradas en forma de vector numérico, por lo que para introducir caracteres será necesario aplicar una codificación. Yo he elegido la codificación *One Hot*, explicada en la sección 2.2.1. A modo de ejemplo, en la Tabla 1 se muestra la codificación de cada una de las vocales para el siguiente vector de salida: [a, e, i, o, u]:

	Codificación				
a	1	0	0	0	0
e	0	1	0	0	0
i	0	0	1	0	0
o	0	0	0	1	0
u	0	0	0	0	1

Tabla 1: Ejemplo de codificación *One Hot*

Como todo problema de *Machine Learning*, es necesario realizar una división del conjunto de datos. Habrá que crear tres subconjuntos diferentes: el de entrenamiento, utilizado para entrenar el modelo; el de validación, utilizado para analizar cómo funciona el modelo y para poder ajustar hiperparámetros; y el de prueba, para medir la precisión del modelo. Es fundamental que los datos de validación y de prueba no se utilicen como entrenamiento, ya que deben de ser desconocidos para la red. Para definir el porcentaje de datos que se incluyen en cada conjunto hay multitud de técnicas y dependerán del objetivo del problema.

Para este trabajo, haré dos divisiones distintas: una que denominaré como "convencional", ya que los porcentajes serán 70%-15%-15% para entrenamiento, validación y test, respectivamente; y otro "reducido", creado para generar *overfitting* en el modelo, ya que, al reducir el conjunto de entrenamiento, es más sencillo para la red recordar los patrones de entrada. Esta última división reducirá el conjunto de entrenamiento en un 92,8%, manteniendo los otros dos conjuntos del mismo tamaño para poder comparar todas las pruebas.

[6] https://gist.github.com/jsdario/6d6c69398cb0c73111e49f1218960f79

	División convencional	División reducida
Entrenamiento	704800	51200
Validación	150400	150400
Test	150400	150400

Tabla 2: Número de caracteres en cada conjunto para las dos divisiones del problema

3.7 Calidad de las soluciones

Para medir la calidad de la solución, tomaré como referencia las métricas de pérdida (*loss*) y precisión (*accuracy*) en el conjunto de validación. La función de coste elegida es la *categorical cross entropy*, también llamada *softmax loss*, que es una generalización de la función *cross entropy* pero para un problema multiclase como el de este trabajo. Está compuesta por una *softmax* seguida de la propia *cross-entropy* que permiten, para una codificación *One Hot*, que únicamente la clase de salida tenga el valor 1 mientras que el resto se quedan a 0.

La precisión mide el porcentaje de salidas correctas que devuelve el modelo para un conjunto determinado. Al ser un problema de predicción con tantas clases, la predicción no va a ser muy alta, por lo que será una métrica secundaria, siendo la pérdida la principal.

3.8 Interpretabilidad de las soluciones

Uno de los efectos conocidos del ruido es polarizar las neuronas, lo que va a facilitar la interpretabilidad de la red [20]. Esta interpretabilidad dependerá de la facilidad para conocer en qué estado de activación se encuentra cada neurona dada una entrada determinada. Es por ello que, para lograr una mejora en la interpretabilidad, los estados posibles de las neuronas se deberían reducir a 1 o -1, como valores extremos de la función de activación; o a 0, cuando la neurona se apague.

Para medir este efecto he utilizado la entropía, que es una medida del desorden de un sistema. Como el orden tiene una entropía menor que el caos, a menor entropía, menos estados distintos en los que se encuentran las neuronas y, por tanto, mayor interpretabilidad de la red. Con esta magnitud, se obtiene el valor máximo cuando una neurona se sitúa siempre en el mismo valor, y el valor mínimo cuando se sitúe en todos los valores posibles el mismo número de veces.

3.9 Keras como elección de desarrollo

Para el desarrollo de este trabajo he optado por utilizar el *software* de desarrollo Keras. Esta elección se debe, por un lado, a que yo ya estaba familiarizado con él, y por otro, a su flexibilidad que permite trabajar tanto a alto nivel, para definir el modelo, como a más bajo nivel, que me ha permitido modificar capas base con los ruidos necesarios para las pruebas.

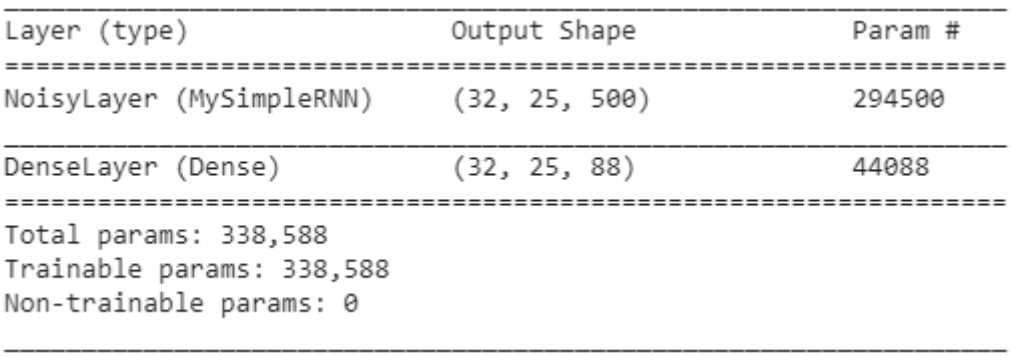


Figura 8: Arquitectura del modelo definido en Keras

Para desarrollar la *Elmann RNN* en Keras habrá que definir una primera capa de *Input*, encargada de recibir los datos y crear el Tensor de entrada; una capa oculta recurrente, la '*NoisyLayer*', que es la capa que adapto para introducir ruido, partiendo de la ya desarrollada por Keras *SimpleRNN* y, por último, una capa de salida de tipo *Dense* con una función de activación *softmax*, necesaria para obtener las probabilidades de cada carácter. En la Fig. 8 se ilustra esta arquitectura gracias al método *summary* de los modelos generados en Keras. La primera capa de *Input* no aparece explícitamente ya que el *software* la crea de manera implícita en la primera capa de la red, en este caso la *'NoisyLayer'*, al definir el tamaño de la entrada.

4 Desarrollo

4.1 Pruebas iniciales

Comenzaré poniendo en práctica lo mencionado en los apartados anteriores. El objetivo va a ser observar el efecto del ruido, aplicando primeramente el ruido independiente debido a su sencillez, en el problema de generación de texto con una división entre conjuntos convencional (apartado 3.6).

Tal y como queda explicado en el apartado 3.1, el modelo cuenta con 3 capas. De estas 3, la de entrada y la de salida deben tener 88 neuronas, debido a la representación en *One Hot* de los 88 caracteres distintos que contempla el problema. La capa intermedia no tiene un número establecido de neuronas por lo que, cuanto mayor sea el número, más crecerá la complejidad del modelo.

Elegiré como optimizador el SGD (apartado 3.4) aplicando los hiperparámetros obtenidos en el Anexo A. Entonces, el primer paso va a consistir en comparar los resultados de variar el número de neuronas en la capa oculta en dos modelos: uno sin ruido y uno con un ruido independiente de valor 1.

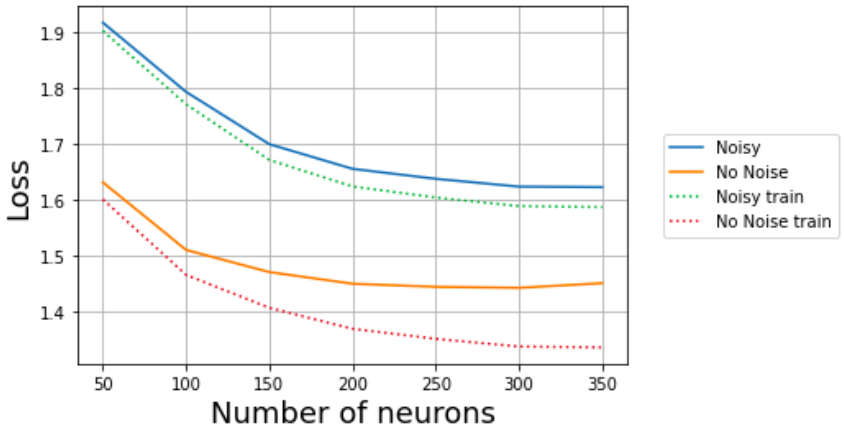


Figura 9: Influencia del ruido en la pérdida al variar el número de neuronas en la capa oculta

Como se puede observar en la Fig.9, la versión sin ruido siempre tiene una pérdida menor en validación para los tamaños de capa contemplados y la pérdida en entrenamiento siempre será menor si observamos la tendencia. Por tanto, el ruido empeora la precisión de la red, no ayuda a mejorar la pérdida en validación sea cual sea el tamaño de la capa recurrente. Hay que mencionar que esta prueba ya es bastante ejemplificante de la actuación del ruido en el modelo, ya que se puede observar que este comienza a tener influencia cuando el problema comienza a sufrir cierto *overfitting*. Es necesario mencionar que el problema que estoy contemplando no sufre mucho sobreajuste, debido a que la complejidad del modelo no es lo suficientemente alta y además el conjunto de datos de entrenamiento es muy extenso.

El siguiente paso consistió en analizar que ocurre con la entropía al introducir ruido y aumentar la complejidad del modelo añadiendo más neuronas en la capa oculta.

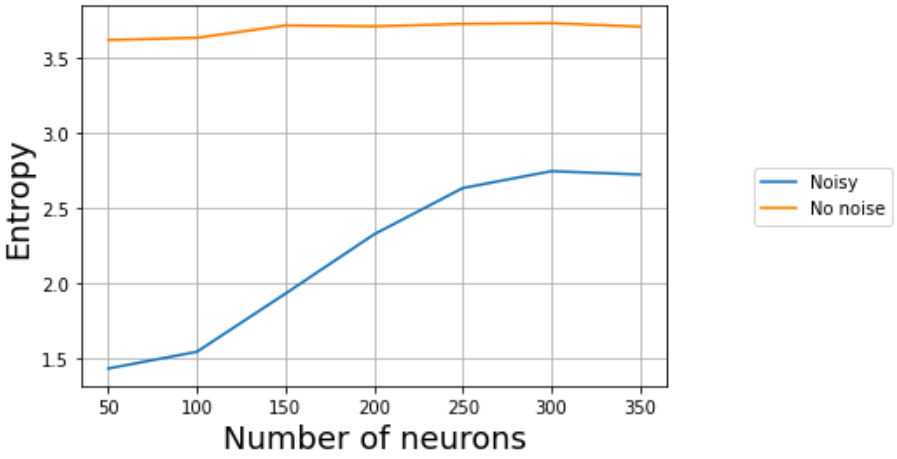


Figura 10: Influencia del ruido en la entropía al variar el número de neuronas de la capa oculta

La Fig.10 muestra que introducir ruido en el modelo supone una reducción de la entropía sea cual sea el número de neuronas de la capa oculta. En este barrido se alcanzó un número máximo de 350 neuronas, que permitía percibir un cierto *overfitting* pero, como he mencionado anteriormente, este problema es pequeño debido a la baja complejidad del modelo y la gran cantidad de datos de entrenamiento. Por tanto, el siguiente paso consistió en situarse en un punto de complejidad algo más alta, con 400 neuronas en la capa oculta, para generar alto más de sobre ajuste y ver que efecto provoca introducir distintos niveles de ruido.

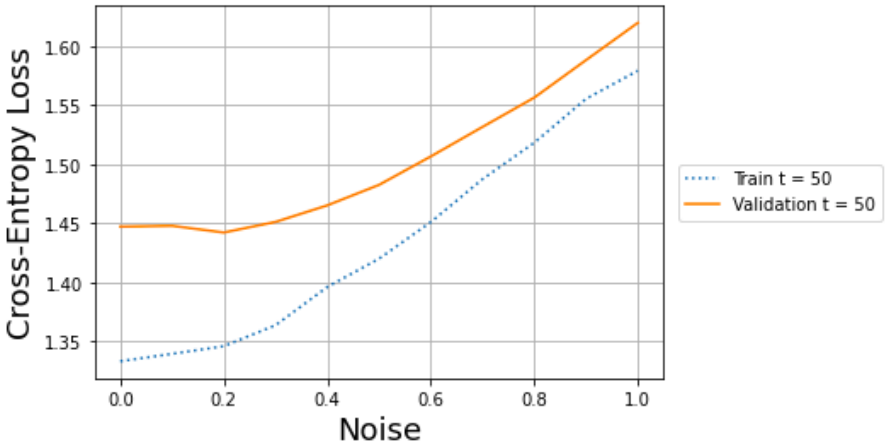


Figura 11: Influencia de diferentes valores de ruido en la pérdida

Como se puede observar en la Fig.11, un ruido más elevado consigue acercar las curvas de evaluación y test, aunque perjudica la pérdida en validación. Lo que consigue el ruido es provocar que las neuronas intenten polarizarse para librarse del ruido, desplazándose a las regiones con derivada 0 de la función *tanh*, es decir, asumiendo valores próximos a -1 o 1. Podemos observar este fenómeno en las siguientes gráficas de activación:

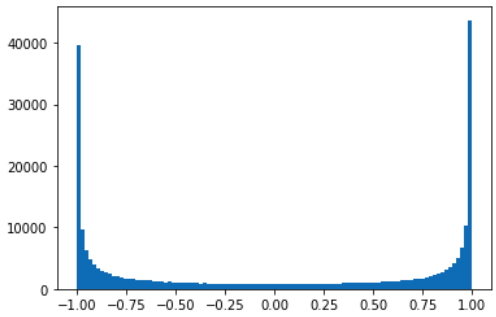


Figura 12: Activaciones de una neurona sin ruido

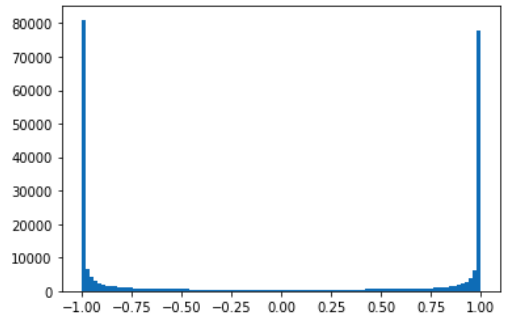


Figura 13: Activaciones de una neurona con ruido

En las Fig. 12 y Fig. 13 se puede observar el fenómeno de polarización anteriormente mencionado. Con el ruido, las neuronas tienden a polarizarse en mayor medida y abandonan los valores intermedios. Por tanto, las primeras pruebas demostraron que para un escenario sin *overfitting* el ruido no ayuda porque no es necesario, pero sí que ayuda a generar el efecto de polarización.

Una vez obtenidos los primeros resultados y observar que realmente el ruido actúa como un regularizador y, por tanto, para problemas sin *overfitting* no es positivo, el siguiente paso consistirá en situarnos en una complejidad alta del modelo, donde se sufre cierto sobreajuste, para establecer las condiciones generales del estudio. Los modelos estarán compuestos por una capa recurrente de 500 neuronas y serán entrenados durante 200 épocas. Esto será común para el resto del documento. Además, el ruido que se introducirá estará comprendido entre 0.0, la versión sin ruido, y 2.0, el ruido máximo y se hará de forma independiente.

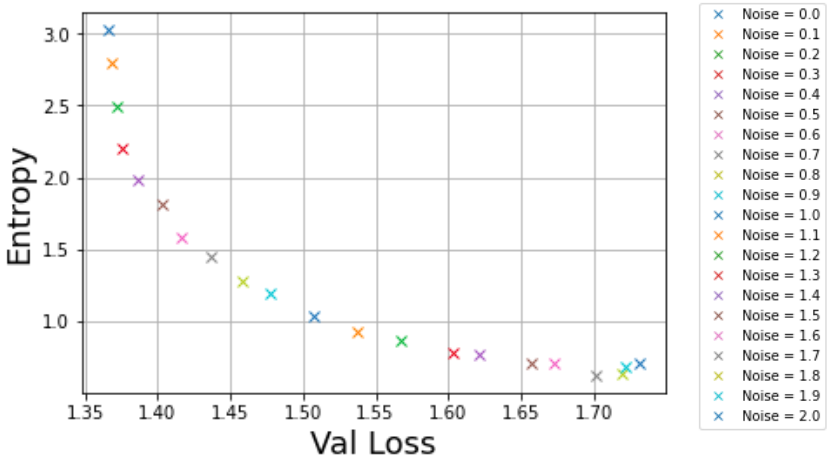


Figura 14: Entropía frente a pérdida para problema con división convencional y ruido independiente

En la Fig. 14 se observa claramente como el ruido no contribuye a mejorar la pérdida en validación, pero sí que mejora la entropía de la red. Se confirma que el modelo sin ruido es el que alcanza una menor pérdida en validación y el modelo con ruido máximo, el que logra peor pérdida. Es interesante observar que con ruidos pequeños (< 0.5) la pérdida no se ve muy afectada y la entropía desciende bastante. Esto significa que, con una mínima introducción de ruido, ya son bastantes más neuronas que se polarizan, pero no llegan a ser

demasiadas como para entorpecer la precisión. El objetivo será acercarse al extremo izquierdo inferior de la gráfica, más cercano a valores de ruido intermedios.

Llegados a este punto, se puede comprobar que el ruido no ha ayudado mucho debido a que no era necesario, ya que el problema con división convencional tiene una gran cantidad de datos de entrenamiento y la complejidad del modelo no es lo suficientemente alta como para generar *overfitting*. El siguiente paso será entonces forzar al problema a que genere sobre ajuste para que el ruido si que sea beneficioso por su labor regularizadora.

4.2 Generando overfitting

Debido a la gran cantidad de datos de entrenamiento que tiene la modalidad "convencional" del problema, el tiempo de entrenamiento es considerablemente alto. Además, observando que el ruido ayuda en contextos donde se produce *overfitting*, voy a utilizar a partir de ahora la modalidad "reducida" del problema, que permitirá realizar mayor cantidad de pruebas y de una manera más ágil.

Las pruebas tendrán las mismas condiciones: 500 neuronas en la capa oculta y 200 épocas de entrenamiento.

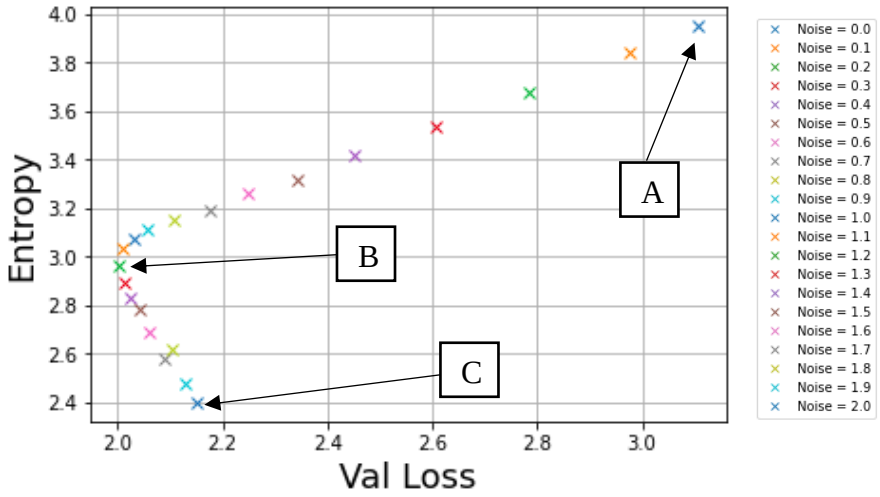


Figura 15: Entropía frente a pérdida para problema con división reducida y ruido independiente

Los resultados obtenidos en la Fig.15 evidencian el *overfitting* que sufre el problema debido a la división de conjuntos planteada. La pérdida en validación se resiente, pero se observa un fenómeno interesante: el ruido ahora sí que ayuda a mejorar la precisión del modelo. La entropía en este caso también se ve mejorada. Se empiezan a observar los primeros beneficios del ruido, mejorando los valores de entropía del caso base, pero aún por encima en pérdida en validación.

Para observar más detalladamente el efecto del ruido en estos modelos, va a ser muy interesante observar las activaciones de las neuronas.

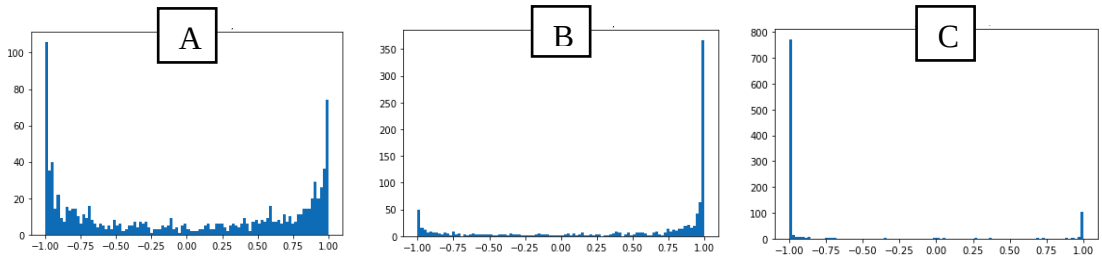


Figura 16: Activaciones con ruido 0.0, 1.2 y 2.0 para problema con división convencional y ruido independiente

La Fig. 16 muestra las activaciones de una neurona elegida al azar en tres supuestos distintos: cuando no se le aplica ruido (A), cuando se le aplica el ruido que hace que alcance el mejor valor de entropía frente a pérdida (B) y con ruido máximo (C). En la versión sin ruido se puede observar que hay muchos valores sin polarizar, en la región central de la figura. A medida que se introduce ruido, los valores se desplazan a los extremos consiguiendo polarizarse, disminuyendo la entropía. Para intentar favorecer este efecto, voy a hacer uso de los regularizadores L1 y L2.

4.3 Influencia de los regularizadores L1 y L2

Los regularizadores (apartado 3.5) penalizan la complejidad del modelo para simplificarlo, perjudicando la tarea de memorización de la red. Este proceso lo realizan disminuyendo los pesos de las neuronas, incluso llegándolas a apagar. Es un efecto contrario al que produce el ruido, pero el objetivo será lograr un punto de convivencia donde la regularización apague las neuronas polarizadas por el ruido y, con todo ello, favorecer la entropía y la reducción del *overfitting*.

El primer paso consistirá en encontrar un valor de regularización L1 que ayude en la tarea., analizando un rango de valores comprendidos entre 1e-4 y 1e-8.

	REGULARIZACIÓN					
Pérdida	1e-4	1e-5	1e-6	1e-7	1e-8	0
Ruido = 0.0	2.15	2.22	3.16	3.11	3.11	3.10
Ruido = 0.2	2.42	2.51	2.90	2.79	2.80	2.78
Ruido = 0.4	2.77	2.89	2.62	2.48	2.47	2.45
Ruido = 0.6	3.21	3.25	2.42	2.26	2.26	2.25
Ruido = 0.8	3.61	3.57	2.32	2.13	2.12	2.10
Ruido = 1.0	3.99	3.85	2.28	2.06	2.03	2.03
Ruido = 1.2	4.25	4.10	2.29	2.03	2.01	2.00
Ruido = 1.4	4.52	4.31	2.31	2.05	2.03	2.02
Ruido = 1.6	4.66	4.49	2.35	2.09	2.06	2.06
Ruido = 1.8	4.84	4.65	2.40	2.13	2.10	2.10
Ruido = 2.0	4.99	4.81	2.44	2.17	2.14	2.15

Tabla 3: Pérdida en validación para diferentes valores de regularización L1 en problema con división reducida y ruido independiente

Entropía	REGULARIZACIÓN					
	1e-4	1e-5	1e-6	1e-7	1e-8	0
Ruido = 0.0	2.66	2.72	4.23	3.97	3.94	3.95
Ruido = 0.2	2.78	2.99	4.01	3.70	3.68	3.68
Ruido = 0.4	3.11	3.21	3.85	3.46	3.41	3.41
Ruido = 0.6	3.39	3.45	3.80	3.32	3.23	3.25
Ruido = 0.8	3.74	3.71	3.83	3.23	3.12	3.15
Ruido = 1.0	3.99	3.93	3.85	3.18	3.07	3.07
Ruido = 1.2	4.15	4.11	3.86	3.13	3.00	2.96
Ruido = 1.4	4.29	4.24	3.85	2.99	2.87	2.82
Ruido = 1.6	4.39	4.31	3.79	2.91	2.78	2.68
Ruido = 1.8	4.47	4.36	3.74	2.81	2.71	2.62
Ruido = 2.0	4.58	4.39	3.71	2.74	2.55	2.40

Tabla 4: Entropía en validación para diferentes valores de regularización L1 en problema con división reducida y ruido independiente

Las Tablas 3 y 4 muestran que el efecto del ruido y de la regularización se contraponen, observable al analizar lo que ocurre con valores de ruido pequeños, donde aumentar la regularización supone una mejora en pérdida y en entropía. Justamente lo contrario ocurre con ruidos altos, donde los valores de pérdida y entropía se mejoran cuanto menor sea el valor de regularización. Si se analiza la tabla por columnas, se observa el mismo efecto, observando que a partir de un valor de 1e-6 ya la regularización es lo suficientemente pequeña para que el ruido tenga mayor influencia. Además, se puede concluir que, si se quiere introducir ruido con esta estrategia, la mejor opción es no regularizar. Los resultados para este valor de regularización se recogen en la Fig. 17:

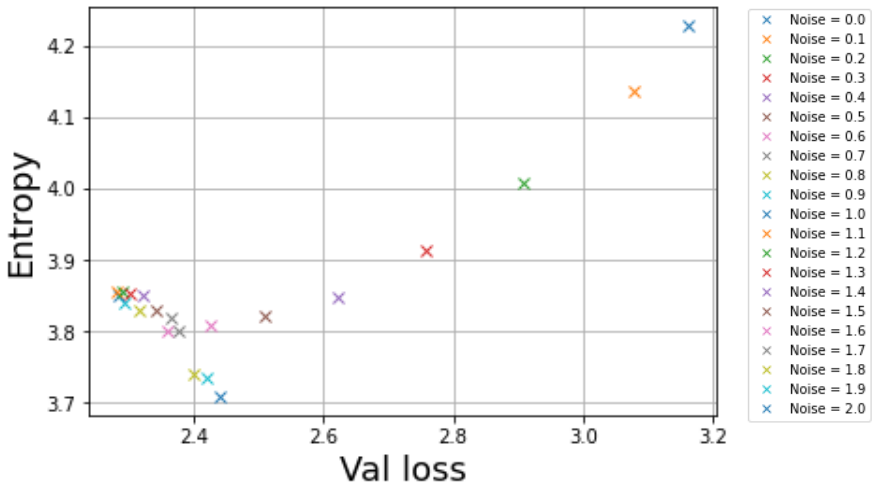


Figura 17: Entropía frente a pérdida para problema con división reducida, ruido independiente y regularización L1 con valor 1e-6

El problema reside en que el ruido introducido, al ser independiente, influye a todas las neuronas sea cual sea su valor de activación. Esto impide que la regularización haga su trabajo, ya que, si consigue apagar neuronas, el ruido puede sacarlas de ese estado. Es por ello que, para intentar que ambos efectos coexistan, voy a utilizar el ruido dependiente al introducir regularización. En este ruido (apartado 3.2.2), las neuronas con valores de

activación cercanos a cero se verán menos influidas por el ruido, llegándose a eliminar el efecto del ruido para neuronas completamente apagadas.

El ruido dependiente sin regularización genera los siguientes resultados:

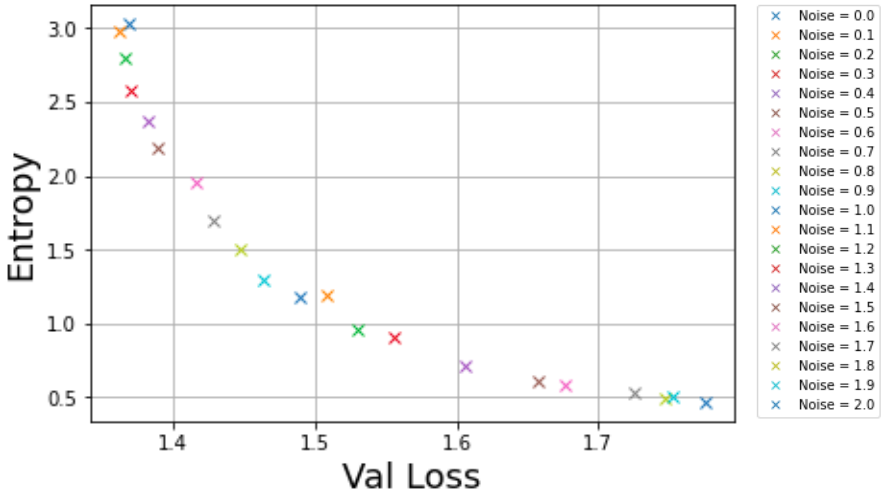


Figura 18: Entropía frente a pérdida para problema con división convencional y ruido dependiente

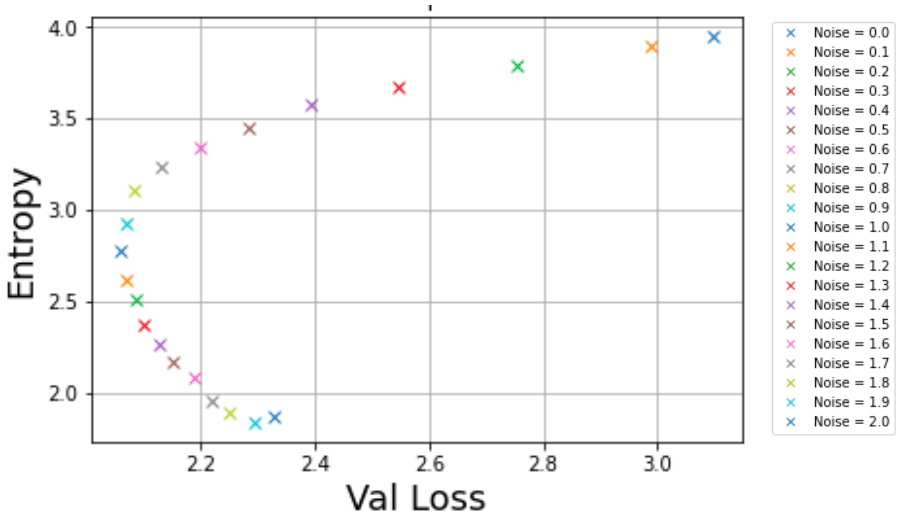


Figura 19: Entropía frente a pérdida para problema con división reducida y ruido dependiente

Esta técnica, ilustrada en las Fig. 18 y 11, es menos agresiva que la modalidad independiente, condición que le perjudica en este caso, ya que sus homólogas independientes obtienen mejores resultados. La diferencia es que las planteo como la solución para que el ruido conviva con la regularización. El siguiente paso será comprobar si para este tipo de ruido es posible la coexistencia.

Comenzaré analizando la regularización L1, buscando el valor que ayuda en combinación con el ruido.

Pérdida	REGULARIZACIÓN					
	1e-4	1e-5	1e-6	1e-7	1e-8	0
Ruido = 0.0	2.32	2.21	3.15	3.11	3.10	3.08
Ruido = 0.2	2.28	2.17	2.59	2.76	2.73	2.76
Ruido = 0.4	2.47	2.19	2.06	2.38	2.38	2.37
Ruido = 0.6	5.20	2.53	2.07	2.19	2.20	2.17
Ruido = 0.8	11.14	3.21	2.22	2.09	2.07	2.06
Ruido = 1.0	16.64	5.67	2.57	2.08	2.05	2.08
Ruido = 1.2	22.01	6.05	2.69	2.13	2.08	2.09
Ruido = 1.4	37.89	6.27	2.76	2.18	2.13	2.17
Ruido = 1.6	38.85	6.24	2.79	2.22	2.18	2.24
Ruido = 1.8	38.88	6.30	2.83	2.28	2.24	2.33
Ruido = 2.0	39.57	6.27	2.87	2.37	2.33	2.35

Tabla 5: Pérdida en validación para diferentes valores de regularización L1 en problema con división reducida y ruido dependiente

Entropía	REGULARIZACIÓN					
	1e-4	1e-5	1e-6	1e-7	1e-8	0
Ruido = 0.0	0.75	2.66	4.23	3.98	3.95	3.96
Ruido = 0.2	0.68	2.30	4.30	3.83	2.78	3.82
Ruido = 0.4	0.75	2.22	4.31	3.64	3.58	2.62
Ruido = 0.6	3.18	3.27	4.28	3.42	3.35	3.45
Ruido = 0.8	3.81	4.14	3.96	3.17	3.14	3.19
Ruido = 1.0	4.31	3.19	2.65	2.88	2.79	2.78
Ruido = 1.2	4.48	2.47	2.15	2.57	2.58	2.57
Ruido = 1.4	2.65	1.19	1.28	2.40	2.32	2.37
Ruido = 1.6	2.35	1.36	2.01	2.11	2.13	2.11
Ruido = 1.8	2.37	1.67	1.26	2.03	1.99	2.04
Ruido = 2.0	1.82	1.99	1.69	1.56	1.62	1.92

Tabla 6: Entropía en validación para diferentes valores de regularización L1 en problema con división reducida y ruido dependiente

Lo más interesante de observar en las Tablas 5 y 6 es cómo actúan ahora ambas técnicas en conjunto. Se observa como ambas técnicas ahora tienen mayor capacidad de coexistir con valores altos de ruido y regularización. Además, ahora si que se observan mejores resultados al introducir para los modelos con cierta regularización baja (1e-7 y 1e-8) con respecto a la versión sin regularizar. Se muestran en la Fig. 20 los resultados de aplicar un valor de regularización 1e-7, con resultados algo peores que 1e-8, pero busco permitir a la regularización tener algo más de influencia.

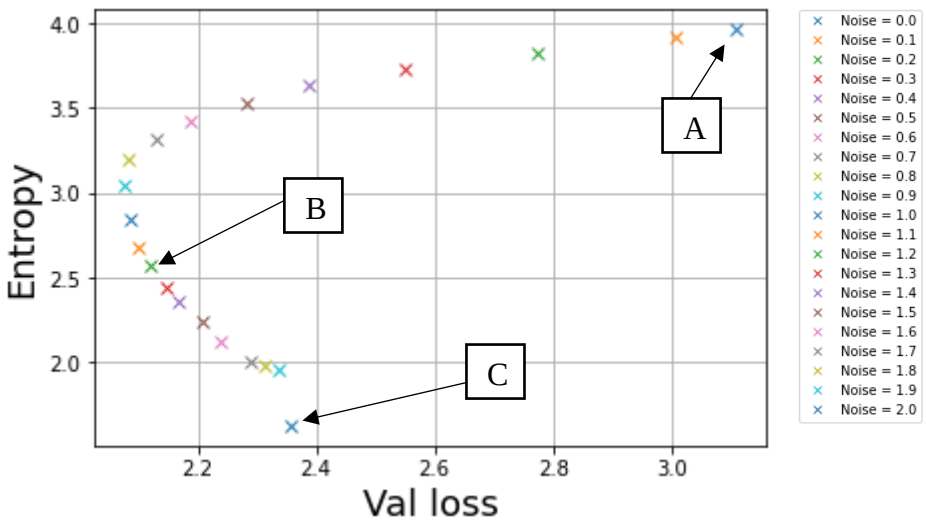


Figura 20: Entropía frente a pérdida para problema con división reducida, ruido dependiente y regularización L1 de valor 1e-7

En la Fig. 20 se observan ligeras mejoras a nivel de entropía para ruidos altos con respecto a las Fig. 19, gracias al efecto de la regularización; y con respecto a la Fig. 17, gracias al cambio de estrategia del ruido.

Observemos el enfrentamiento entre las técnicas de regularización L1 y ruido en las activaciones de las neuronas:

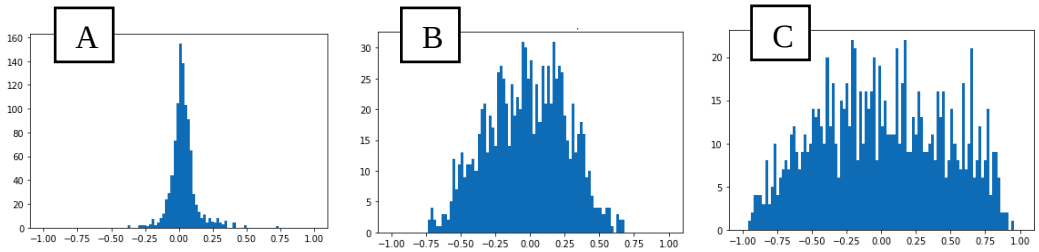


Figura 21: Activaciones con regularización L1 de 1e-5 y ruidos dependientes de valor 0.0, 1.2 y 2.0

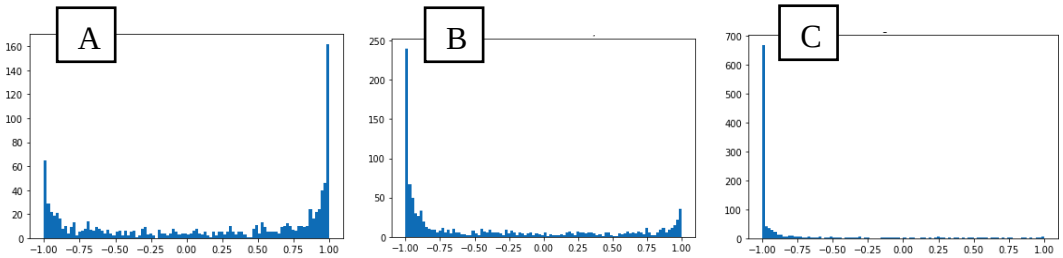


Figura 22: Activaciones con regularización L1 de 1e-7 y ruidos dependientes de valor 0.0, 1.2 y 2.0

La Figura 21 muestra las neuronas cuando son afectadas por una regularización L1 alta (1e-5). Se observa cómo se minimizan los pesos, acumulándose las activaciones en valores centrales, consiguiendo simplificar el modelo apagando neuronas, pero afectando negativamente a la entropía. La introducción de ruido en este caso intenta desplazar a valores

algo más extremos, por eso se puede apreciar el cambio a una forma menos afilada. Por otro lado, en la Figura 22 observamos lo contrario: la regularización consigue apagar los valores centrales y el ruido, al tener mayor poder en este caso, es capaz de llevarse las activaciones a los valores límite.

Tal y como he mencionado en el apartado 3.5, la regularización L1 es más afectiva que la L2 para lograr una mayor interpretabilidad debido a su mayor agresividad apagando neuronas. Para demostrar este efecto, probaré el mismo rango de valores de regularización ahora para el tipo L2:

	REGULARIZACIÓN					
Pérdida	1e-4	1e-5	1e-6	1e-7	1e-8	0
Ruido = 0.0	2.14	2.05	3.18	3.11	3.10	3.10
Ruido = 1.2	13.66	3.42	2.22	2.13	2.02	2.09
Ruido = 2.0	21.01	4.24	2.40	2.37	2.16	2.31

Tabla 7: Pérdida en validación para diferentes valores de regularización L2 para problema con división reducida y ruido dependiente

	REGULARIZACIÓN					
Entropía	1e-4	1e-5	1e-6	1e-7	1e-8	0
Ruido = 0.0	2.44	3.46	4.21	3.98	3.97	3.94
Ruido = 1.2	4.05	4.14	4.02	2.57	3.15	2.47
Ruido = 2.0	4.15	4.40	3.90	1.56	2.94	1.92

Tabla 8: Entropía en validación para diferentes valores de regularización L2 para problema con división reducida y ruido dependiente

Como se puede observar en las Tablas 7 y 8, ocurre el mismo fenómeno que con la aplicación de la regularización L1: una regularización alta junto con ruido empeora el modelo, tanto en pérdida en validación como en entropía. A medida que la regularización decrece, se observa un giro en la influencia del ruido, haciendo que el modelo mejore, pero no se consiguen mejores resultados que en la versión sin regularización. Por tanto, se confirma que la regularización L1 sí que supone cierta ayuda en la tarea de mejorar la interpretabilidad, pero la L2 no.

En definitiva, la regularización y el ruido no funcionan bien juntos debido a que tienen efectos opuestos. La regularización ayuda en ausencia del ruido y viceversa. La aplicación del ruido dependiente supone una mejora en la coexistencia entre ambos, pero los resultados aplicando los dos efectos no son mejores que aplicando únicamente uno de ellos.

4.4 Otros tipos de ruido

Tras analizar los dos primeros tipos de ruido y la influencia de la regularización, el siguiente paso consistirá en probar el resto de las estrategias, comenzando por el ruido posterior a la función de activación (apartado 3.2.3). Esta técnica ofrece una mayor agresividad que la versión anterior, lo que puede ser un arma de doble filo ya que, por un lado, puede provocar que neuronas polarizadas salgan de este estado con mayor probabilidad o, por el contrario,

neuronas en la región derivable pueden ser desplazadas más fácilmente a los extremos. Lo que ocurre realmente, gracias a la forma de la función de activación *tanh*, es que el ruido posterior favorece el desplazamiento hacia valores en los extremos. Este efecto se puede observar en la Fig. 23:

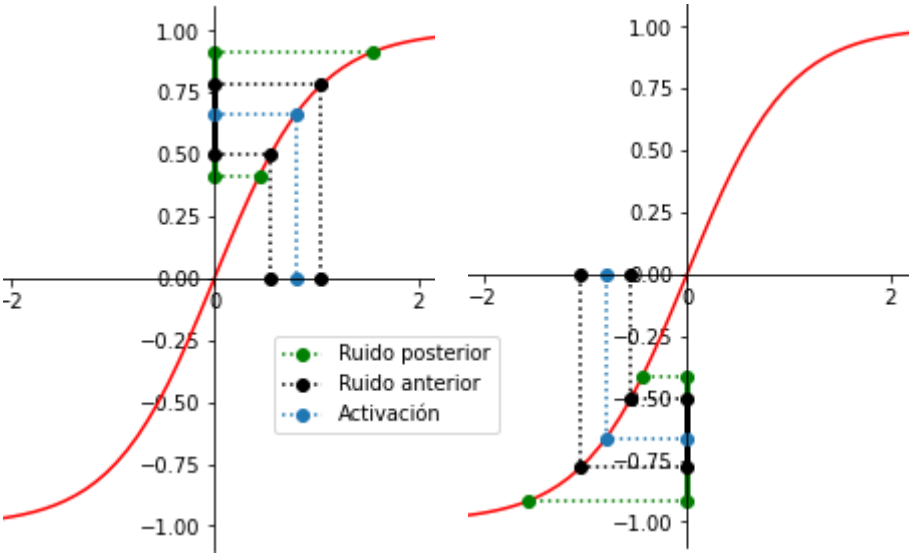


Figura 23: Comparativa entre ruido anterior y posterior

Se puede observar como a medida que la activación se aproxima a valores de polarización, aplicando el mismo valor de ruido, la diferencia entre introducirlo antes o después de la función de activación se agrava. Este efecto puede ayudar en la tarea de la interpretabilidad (ver apartado 3.8), ya que neuronas que estaban más lejanas a la región no derivable ahora tienen mayor probabilidad de polarizarse. Por el otro lado, también se puede conseguir el efecto contrario, sacando neuronas polarizadas de su estado. Observemos su efecto:

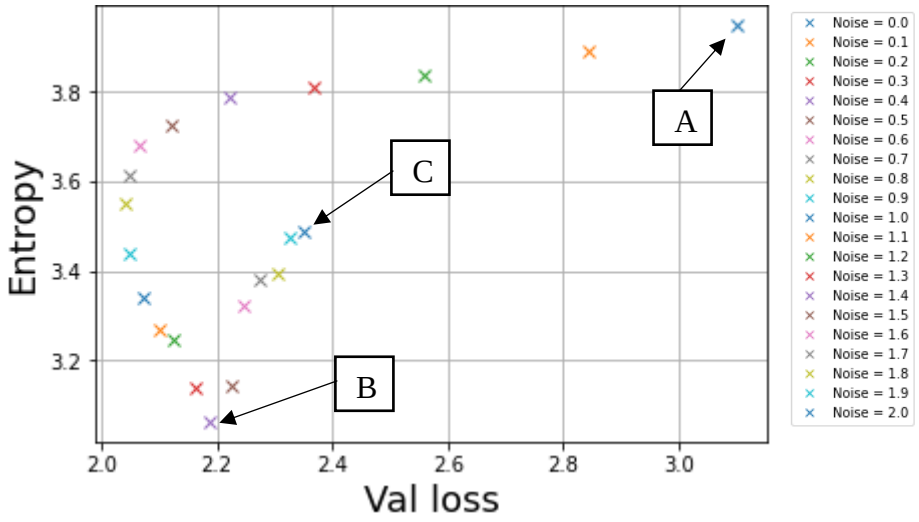


Figura 24: Entropía frente a pérdida para problema con división reducida y ruido posterior

Como se puede observar, los resultados obtenidos no son muy positivos ya que, aunque la pérdida en validación no sea muy negativa, la entropía es alta comparada con pruebas

anteriores. La curva de la Figura 24 indica que el ruido mejora los resultados hasta un valor de 1.4, punto a partir del cual vuelve a la tendencia ascendente. Esto se debe a que, al ser una técnica más agresiva, el aumento del ruido conlleva una mayor inestabilidad de las neuronas. Observemos el efecto en las activaciones:

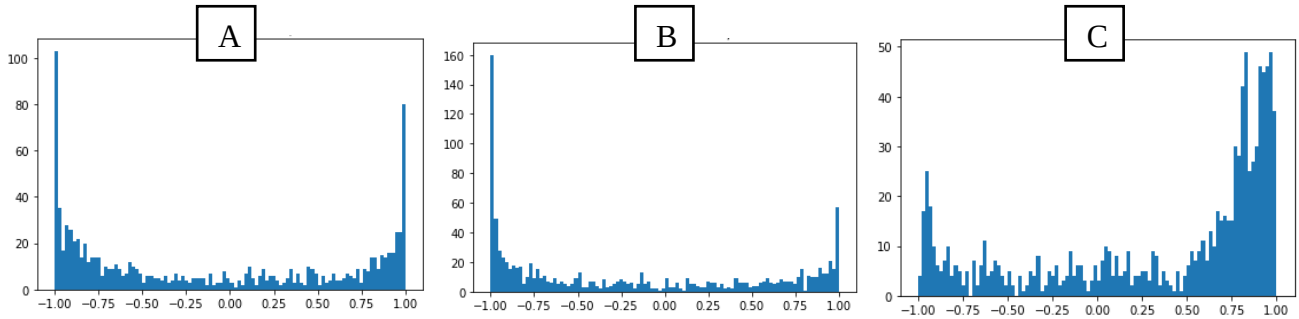


Figura 25: Activaciones de las neuronas para ruidos posteriores de valor 0.0, 1.2 y 2.0

Los casos con ruido 0.0 (A) y 1.4 (B) de la Fig. 25 siguen la misma línea de las pruebas anteriores, el ruido elimina los valores intermedios para llevarlos a los extremos. La diferencia reside en el caso con ruido máximo (C), donde se puede observar una gran inestabilidad en la que, debido al efecto más agresivo explicado anteriormente, los valores se acumulan cerca de los valores extremos. Por tanto, se puede concluir que esta estrategia para introducir ruido no es positiva para el objetivo deseado en esta investigación.

Por último, la estrategia que voy a probar es introducir ruido dependiente del estado actual de la neurona. Esta técnica tiene como objetivo ser más agresiva, ya que, al utilizar una red recurrente, el estado actual depende del estado y estados anteriores y por esto es lo que funciona la primera estrategia utilizada. Al atacar directamente el estado actual, se puede influir de forma más directa y provocar la polarización o la desconexión.

La diferencia sustancial con introducir ruido dependiente del estado anterior es que el valor de recursión ya está acotado por la función de activación, por tanto, a pesar de que el nivel de ruido sea grande, la aportación también estará acotada. Esto no ocurre en este caso, ya que el estado actual se obtiene previo a la aplicación de la función de activación, por lo que ruidos muy altos junto con un estado de activación grande generarán mucha inestabilidad.

Observemos cómo influye el ruido para este caso:

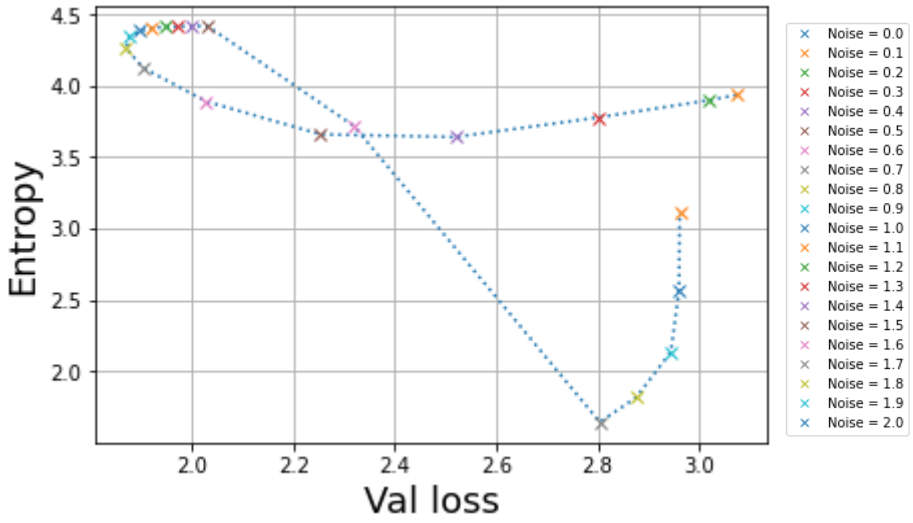


Figura 26: Entropía frente a pérdida para problema con división reducida y ruido dependiente del estado actual v1

En la Fig. 26 se unen los puntos resultantes para permitir observar más claramente la tendencia que sigue la figura y, cómo ruidos pequeños, favorecen la pérdida en validación sin mejorar la entropía, debido a la inestabilidad que se genera, porque el ruido tiene mucha más influencia que en casos anteriores. Llegados a un valor de ruido de 1.5, se produce una caída importante en la entropía, debido a que muchas neuronas se apagan, lo que perjudica a la pérdida. A partir de ese nivel de ruido, su influencia y la inestabilidad que genera es tan grande que la entropía crece siguiendo una tendencia ascendente prácticamente en vertical.

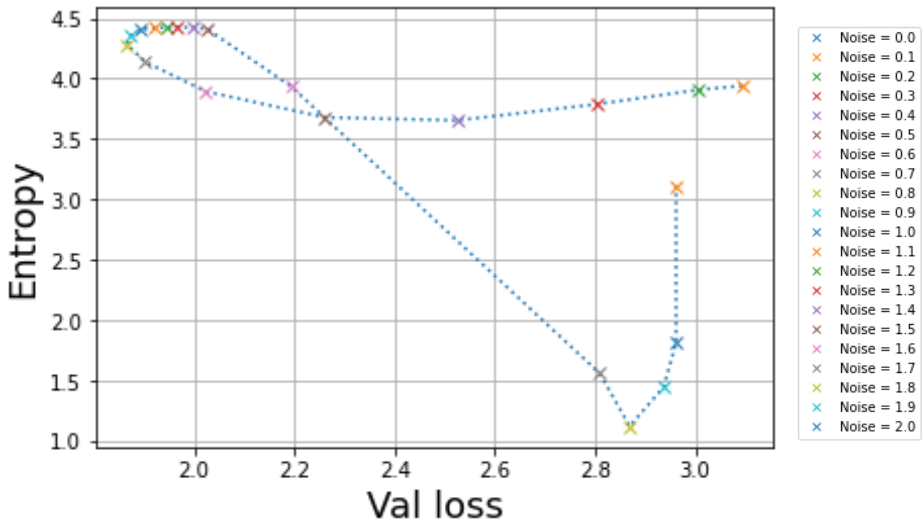


Figura 27: Entropía frente a pérdida para problema con división reducida, ruido dependiente del estado actual v1 y regularización L1 de valor 1e-7

Si introduzco regularización L1 al modelo, se mantiene la forma de la Fig. 27 pero se intensifica el efecto de apagar neuronas con la consecuencia de que la entropía disminuirá aún más. A pesar de esto, los resultados no son mejores que los modelos anteriores porque, aunque la pérdida para ruidos pequeños es baja, la entropía es muy alta. Justamente lo contrario que ocurre para ruidos más elevados.

Para reducir la influencia que tiene el ruido y que no perjudique tanto para valores altos, la siguiente técnica hace que el estado actual esté acotado entre 1 y -1. Esto se consigue calculando el valor de la neurona previo paso por la función de activación correspondiente, la *tanh* en mi caso. Ese valor será el utilizado para acotar el ruido introducido. Se explica con mayor detalle en el apartado 3.2.6. En este caso, tanto la conexión recurrente como el estado actual van a estar acotados entre 1 y -1, lo que reducirá la inestabilidad:

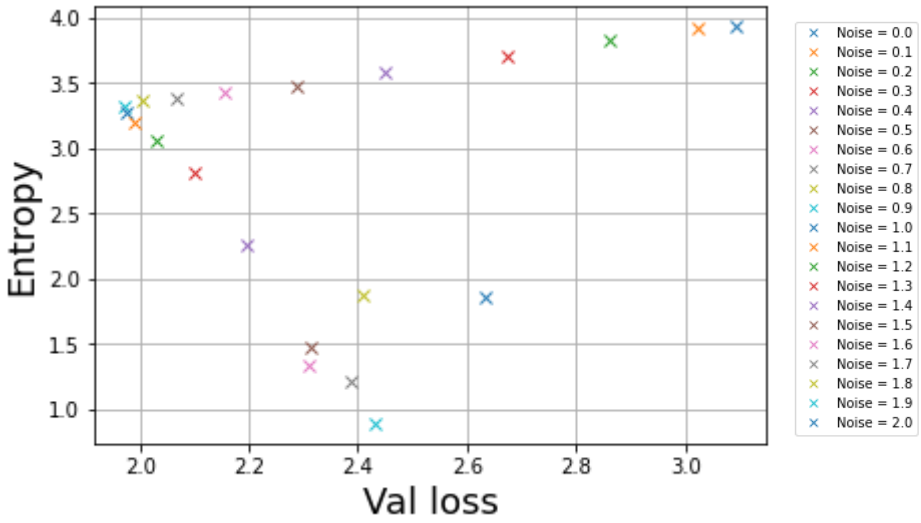


Figura 28: Entropía frente a pérdida para problema con división reducida y ruido dependiente del estado actual v2

La forma de la Fig. 28 se asemeja mucho más a las anteriores, donde la influencia del ruido también estaba acotada. La introducción de valores de ruido elevados genera una mejora en la entropía, sin perjudicar mucho la pérdida en validación. Principalmente se ven beneficiados los valores altos de ruido. Viendo que son los mejores resultados hasta el momento, voy a buscar si existe algún valor de regularización L1 que los mejore:

	REGULARIZACIÓN					
Pérdida	1e-4	1e-5	1e-6	1e-7	1e-8	0
Ruido = 0.0	2.32	2.21	3.13	3.13	3.15	3.10
Ruido = 0.2	2.27	2.17	2.81	2.88	2.90	2.86
Ruido = 0.4	2.29	2.16	2.31	2.47	2.48	2.45
Ruido = 0.6	2.31	2.15	2.02	2.17	2.15	2.14
Ruido = 0.8	2.33	2.17	1.99	2.03	2.05	2.01
Ruido = 0.9	2.35	2.19	2.00	1.98	1.96	1.96
Ruido = 1.0	2.38	2.23	2.03	1.99	1.98	1.97
Ruido = 1.2	2.45	2.28	2.09	2.06	2.01	2.03
Ruido = 1.4	2.51	2.30	2.75	2.31	2.28	2.19
Ruido = 1.6	2.57	6.24	2.74	2.36	2.33	2.31
Ruido = 1.8	38.79	6.34	2.84	2.43	2.42	2.41
Ruido = 2.0	39.65	6.44	2.87	2.51	2.58	2.63

Tabla 9: Pérdida en validación para diferentes valores de regularización L1 para problema con división reducida y ruido dependiente del estado actual v2

Entropía	REGULARIZACIÓN					
	1e-4	1e-5	1e-6	1e-7	1e-8	0
Ruido = 0.0	0.75	2.71	4.24	3.97	3.95	3.93
Ruido = 0.2	0.67	2.40	4.24	3.86	3.84	3.83
Ruido = 0.4	0.63	2.21	4.32	3.63	3.60	3.58
Ruido = 0.6	0.61	2.05	4.25	3.51	3.49	3.41
Ruido = 0.8	0.59	1.95	4.07	3.49	3.40	3.35
Ruido = 0.9	0.60	1.89	3.98	3.48	3.44	3.31
Ruido = 1.0	0.62	1.87	3.95	3.41	3.32	3.26
Ruido = 1.2	0.64	1.84	3.96	3.06	2.43	2.05
Ruido = 1.4	0.65	1.77	1.47	2.26	2.27	2.25
Ruido = 1.6	0.63	1.95	1.04	1.32	1.36	1.33
Ruido = 1.8	0.72	1.82	1.67	1.16	1.56	1.87
Ruido = 2.0	0.76	1.38	1.69	1.81	1.78	1.85

Tabla 10: Entropía para diferentes valores de regularización L1 para problema con división reducida y ruido dependiente del estado actual v2

Gracias a las Tablas 9 y 10 se puede concluir que los mejores resultados se obtienen con regularización alta, de 1e-4. Como se puede observar, para este caso la regularización tiene mayor influencia que el ruido, observado en la escasa variación en los resultados que genera el aumento de este, aunque sí que se observa una cierta mejora para modelos algo ruidosos. Un escenario como este, donde la pérdida en validación no es muy baja pero la entropía sí que lo es, se debe a que muchas neuronas se han apagado, simplificando el modelo, pero sin mejorar la pérdida. Los resultados de la aplicación del regularizador L1 con valor 1e-4 se reflejan en la Fig. 29.

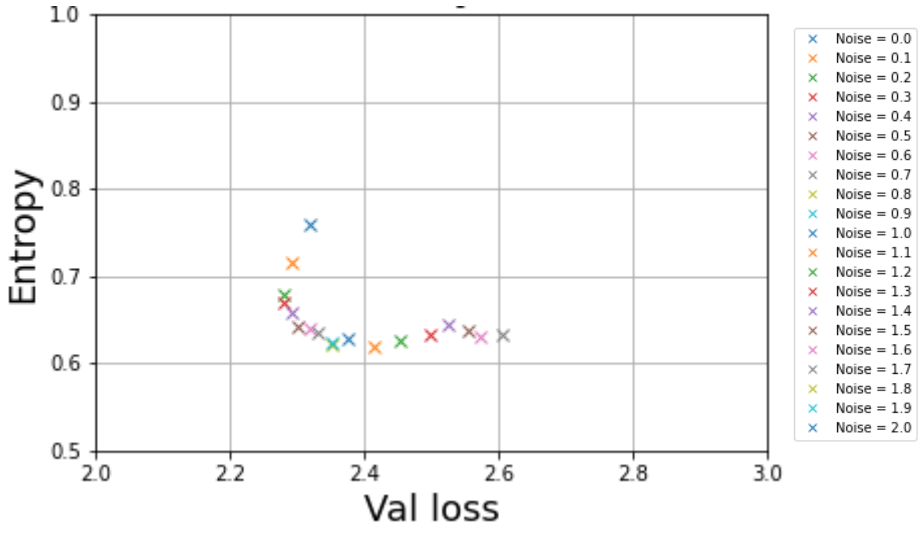


Figura 29: Entropía frente a pérdida para problema con división reducida, ruido dependiente del estado actual v2 y regularización L1 de valor 1e-4

5 Resultados

Esta investigación ha aportado una visión de los efectos del ruido aplicados sobre funciones de activación en redes neuronales recurrentes. Tras todas las pruebas realizadas con las distintas estrategias para introducir el ruido, es interesante conocer cuál de ellas ha conseguido mejores resultados para el problema planteado. Como el objetivo es minimizar tanto la pérdida como la entropía, he elegido como métrica la media armónica, que contempla ambos valores a la vez. Esta métrica se incluye en futuras tablas en la columna 'SCORE'.

VERSIÓN	RUIDO	PÉRDIDA	ENTROPÍA	SCORE	REG
Versión sin ruido	0.0	3.10	3.94	3.47	-
Ruido anterior independiente	2.0	2.16	2.27	2.21	-
Ruido anterior dependiente	1.9	2.29	1.81	2.02	-
Ruido anterior dependiente con regularización L1	2.0	2.35	1.76	2.02	1e-7
Ruido posterior independiente	1.0	2.07	3.31	2.55	-
Ruido dependiente del estado actual v1	1.6	2.79	2.08	2.38	-
Ruido dependiente del estado actual v2	1.9	2.43	0.89	1.31	-
Ruido dependiente del estado actual v2 con regularización L1	0.5	2.29	0.62	0.98	1e-4

Tabla 11: Comparativa de score entre las distintas estrategias de ruido

La Tabla 11 muestra el nivel de ruido, donde el ruido 0 también es una posibilidad, que consigue optimizar el score para cada una de las estrategias. Se observa que, para cada una de ellas, el modelo que optimiza la métrica es un modelo entrenado con un cierto valor de ruido. Además, todas ellas mejoran a la versión sin ruido.

En el inicio de este trabajo, en el apartado 1.2, me propuse tres objetivos que analizaré en los siguientes apartados.

5.1 Objetivo 1: Mejora de la generalización

En el apartado 4.1, se muestran los efectos del ruido y cómo este consigue acercar las curvas de validación y entrenamiento en el problema con división convencional. Una vez reducido el conjunto de entrenamiento, el *overfitting* es mucho más evidente (apartado 4.2) y los efectos del ruido para mejorar la generalización se pueden observar con mayor claridad.

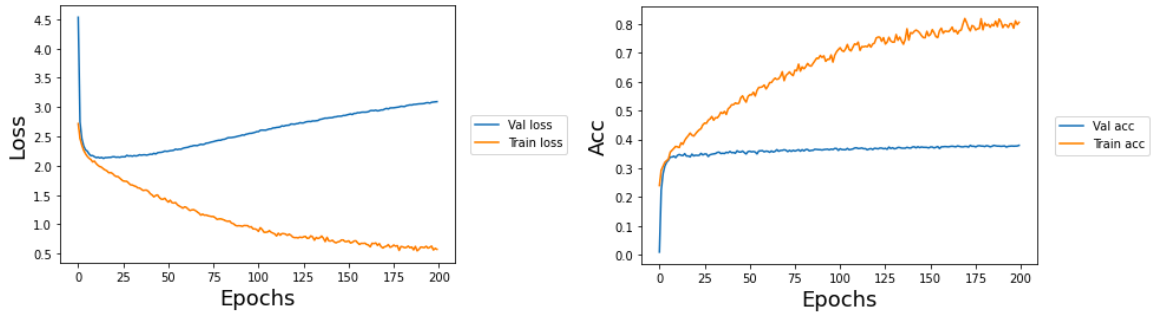


Figura 30: Pérdida y precisión por época para problema con división reducida y sin ruido

La Fig. 30 muestra tanto la pérdida como la precisión en los conjuntos de entrenamiento y validación para el problema con su versión reducida cuando no se introduce ruido. El *overfitting* se observa debido a que las curvas de entrenamiento tienen una tendencia a reducir la pérdida y aumentar la precisión, pero con las curvas del conjunto de validación no ocurre lo mismo. Llegado a un número de épocas, la precisión en validación se estanca y la pérdida aumenta, debido a que la red pierde capacidad de generalización.

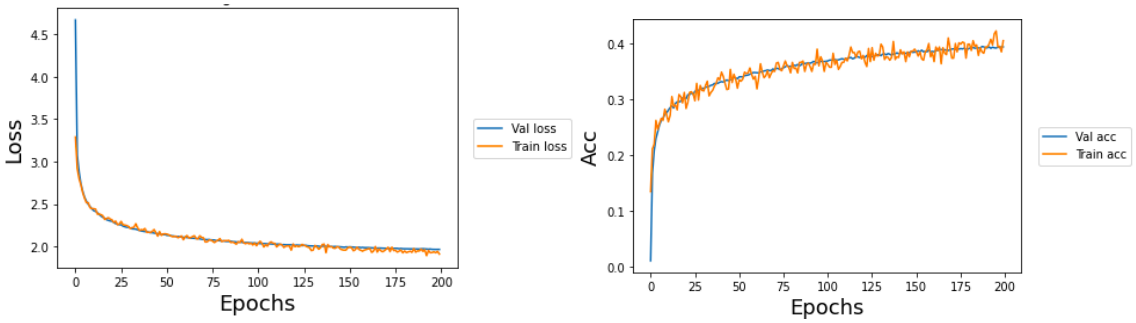


Figura 31: Pérdida y precisión por época para problema con división reducida y con ruido de valor 1.2

La Fig. 31 muestra lo que ocurre al introducir un nivel de ruido de valor 1.2 de tipo dependiente del estado actual v2. El *overfitting* desaparece, ya que las curvas de entrenamiento y validación siguen la misma tendencia. La introducción de niveles de ruido mayores provoca mayor inestabilidad en las curvas de entrenamiento, pero sigue sin producirse el *overfitting*, ejemplificado en la Fig. 32, donde se introduce ruido con valor 2.0.

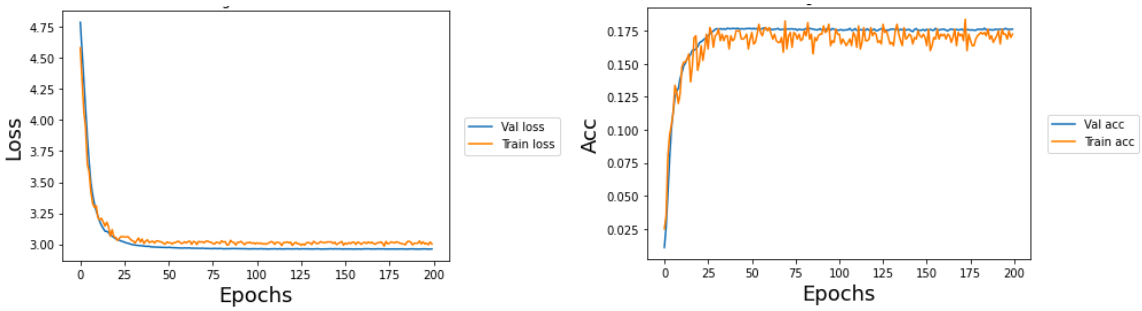


Figura 32: Pérdida y precisión por época para problema con división reducida y con ruido de valor 2.0

Por último, la Tabla 12 ilustra los modelos con menor pérdida en validación para cada estrategia de ruido. En ella se puede observar que todos los modelos óptimos se han entrenado con ruido, por tanto, el ruido ha ayudado a mejorar la pérdida en validación. Además, para todas se obtienen mejores resultados que la versión sin ruido.

VERSIÓN	RUIDO	PÉRDIDA	ENTROPÍA	SCORE	REG
Versión sin ruido	0.0	3.10	3.94	3.47	-
Ruido anterior independiente	1.2	2.00	2.98	2.39	-
Ruido anterior dependiente	1.0	2.06	2.83	2.38	-
Ruido anterior dependiente con regularización L1	0.8	2.06	3.19	2.50	1e-7
Ruido posterior independiente	0.8	2.04	3.53	2.59	-
Ruido dependiente del estado actual v1	0.7	1.86	4.28	2.59	-
Ruido dependiente del estado actual v2	0.9	1.97	3.31	2.47	-
Ruido dependiente del estado actual v2 con regularización L1	0.2	2.27	0.69	1.05	1e-4

Tabla 12: Comparativa de pérdida en validación entre las distintas estrategias de ruido

5.2 Objetivo 2: Mejora de la interpretabilidad

La interpretabilidad de la red la he representado con la de la entropía, ya que en cuantos menos estados de activación se encuentre una neurona, menor entropía habrá y más fácil será modelarlo. La Tabla 13 muestra el modelo con menor entropía para cada estrategia de introducción de ruido, donde se observa algo similar a las Tablas 11 y 12, donde todos los modelos que optimizan la entropía han sido entrenados con ruido y, además, mejoran a la versión sin ruido.

VERSIÓN	RUIDO	PÉRDIDA	ENTROPÍA	SCORE	REG
Versión sin ruido	0.0	3.10	3.94	3.47	-
Ruido anterior independiente	2.0	2.16	2.27	2.21	-
Ruido anterior dependiente	1.9	2.29	1.81	2.02	-
Ruido anterior dependiente con regularización L1	2.0	2.35	1.76	2.02	1e-7
Ruido posterior independiente	1.4	2.19	3.06	2.56	-
Ruido dependiente del estado actual v1	1.6	2.79	2.08	2.38	-
Ruido dependiente del estado actual v2	1.9	2.43	0.89	1.31	-
Ruido dependiente del estado actual v2 con regularización L1	1.2	2.46	0.62	0.99	1e-4

Tabla 13: Comparativa de entropía entre las distintas estrategias de ruido

5.3 Objetivo 3: Aplicación a un problema de generación de texto

El tercer objetivo de este trabajo es aplicar los modelos en el problema de generación de texto planteado en el apartado 3.6. Para poder obtener algo legible, es necesario utilizar los modelos entrenados con la división convencional del problema, es decir, con el conjunto de entrenamiento en su totalidad. Como para este problema el ruido no ayuda en reducir la pérdida, los mejores resultados los obtengo con ruido 0. La mejora de la entropía no es algo observable en este objetivo.

A continuación, se muestra la generación de 500 caracteres, iniciándose con una 'A', valor a partir del cual el modelo comienza a predecir los sucesivos:

"*Ahos paras los caminas, que Don Quijote se decía; pero qué en aquella caballerizo y cantar, una desgraciala que en ello.*
No hay más se hizo en li tardo ni caballero, quien tengo dicho Sancho Panza, sabe boca el volvería a Ron hiciendo del ir,endo, pues la vino más atentor agujero; estores que de la ermicio.
Cerdido Pasó, y decían en adera la lanza acometido, y en algún títrise los Hiulater a donte fuego en el ducil. Callancias con ellas senese, niego de cata, y echando noche a nosía, qu"

Como se puede observar, al no hacer un trabajo de limpieza de datos (apartado 3.6), el modelo es capaz de utilizar letras acentuadas, espacios, comas e incluso saltos de línea. El resultado no es correcto ni gramaticalmente ni sintácticamente, pero sí que se demuestra un cierto aprendizaje de la estructura morfológica del castellano.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Esta investigación ha aportado un análisis sistemático de la influencia del ruido aplicado sobre funciones de activación en redes neuronales recurrentes. He comprobado que el ruido es un regularizador y, por tanto, lo que provoca es penalizar la complejidad del modelo haciéndolo más sencillo. La forma en la que lo consigue es generando inestabilidad a las neuronas cuando se sitúan en la región central de la función de activación, lo que las va a llevar a desplazarse a las zonas más estables, es decir, las regiones donde la derivada es cercana a 0. Las neuronas que se sitúen en estas regiones adoptarán valores muy cercanos a 1 o -1, por lo que el modelo se simplificará.

Debido a su condición de regularizador, el ruido no ayuda en un problema que no sufre de *overfitting*, porque no es necesario. Es por lo que, para la división convencional del problema, que cuenta con muchos datos de entrenamiento y no tiene la suficiente complejidad, se observa que la introducción de ruido supone empeorar los resultados de pérdida en validación, aunque sí que consigue reducir las distancias entre las curvas de perdida para ambos conjuntos, reduciendo el mínimo *overfitting* que pueda existir. Lo que también consigue es reducir la entropía del modelo, métrica que he utilizado para tener una intuición de su interpretabilidad.

Para observar realmente los efectos del ruido, es necesario generar *overfitting*. En mi caso, lo provoqué reduciendo el tamaño del conjunto de entrenamiento y aumentando la complejidad del modelo. La introducción de ruido independiente, que se aplica previamente a la función de activación y es la estrategia más sencilla, logra que la pérdida en validación mejore y, además, consigue reducir la entropía de los modelos haciendo que las neuronas abandonen los valores centrales de la función de activación y se desplacen hacia los extremos.

Los regularizadores L1 y L2, reducen la complejidad del modelo aplicando penalización directamente en los pesos de las neuronas, consiguiendo reducir sus valores de activación hasta llegarlas a apagar. El ruido, desplazando neuronas a los valores extremos, y la regularización, apagando neuronas, podrían ser aplicados de manera conjunta para lograr mejorar la interpretabilidad de la red. Para esta tarea resulta mucho más útil la regularización L1, ya que tiene un efecto más agresivo que la L2.

Pero este trabajo ha demostrado que ambos efectos no pueden convivir. La regularización consigue sacar neuronas polarizadas de su estado y el ruido activa neuronas que la regularización había apagado. Es por ello que, en un intento de lograr un escenario de convivencia, apliqué el ruido dependiente. Este tipo de ruido depende del estado anterior en el que se encontraba la neurona, por lo que neuronas ya apagadas no sufrían ruido. Los resultados reflejaron una leve mejora, sobre todo en capacidad de coexistencia de ambos efectos, pero los resultados no han sido mejores que las versiones sin regularizar.

Introducir ruido posterior a la función de activación supone una mayor agresividad, sobre todo en valores más cercanos a los extremos. Este efecto no es positivo ya que, aunque consigue polarizar neuronas que estaban más alejadas de la región de derivada 0, también provoca que neuronas polarizadas salgan de ese estado. Por último, el ruido dependiente del

estado actual de la neurona logra tener una mayor influencia, sobre todo en la versión 2 que he utilizado, situándolo como la estrategia óptima, para minimizar tanto pérdida en validación como entropía, dentro de las analizadas en este trabajo.

6.2 Trabajo futuro

Los objetos de estudio que se plantean para el futuro son los siguientes:

1. Aplicar otros tipos de ruido. En este trabajo he analizado únicamente el ruido introducido en las funciones de activación y generado por una distribución normal, pero existen muchas otras estrategias que varían en el lugar de introducción, la forma de generarlo o las dependencias con otros componentes de la red.
2. Utilizar otro tipo de redes. En este trabajo he utilizado como modelo de referencia las *Elmann RNN* debido a su sencillez. Se podría analizar la influencia del ruido en otros modelos recurrentes, tales como LSTMs o GRUs.
3. Encontrar otra métrica para la interpretabilidad. En este trabajo he utilizado la entropía para medir la interpretabilidad. Aunque es útil, no mide precisamente lo que estoy buscando. La entropía adquiere su valor mínimo cuando una neurona únicamente se sitúa en un estado, lo que a nivel práctico significa que la neurona es inútil, ya que da igual la entrada que reciba la red, la salida de esa neurona será siempre la misma. La métrica objetivo sería aquella que adquiriese su valor mínimo cuando una neurona se sitúe exactamente el mismo número de veces en cada uno de los valores extremos, es decir, en 1 o -1.

7 Referencias

[1] M. Lesno, V. YaLin, A. Pinkus y S. Schoncken, «Multilayer feedforward networks with a nonpolynomial activation function can approximate any function,» 2005.

[2] H. N. T. Y. J. M. B. Han, «Regularizing Deep Neural Networks by Noise,» 2017.

[3] L. Lago-Fernández y C. Oliva, «On the Interpretation of Recurrent Neural Networks ad Finite State Machines,» *ICANN,* pp. 312-323, 2019.

[4] Y. Bengio, «Estimating or Propagating Gradients Through Stochastic Neurons,» 2013.

[5] I. Sutskever, J. Martens y G. Hinton, «Generating Text with Recurrent Neural Networks,» 2011.

[6] W. Pitts y W. McCulloch, «A logical calculus of the ideas immanent in nervous activity,» 1943.

[7] F. Rosenblatt, «The perceptron: A probabilistic model for information storage and organization in the brain,» 1958.

[8] B. Widrow y M. Lehr, «Artificial neural networks of the perceptron, madaline, and backpropagation family» 1993.

[9] D. Rumelhart, G. Hinton y R. Williams, «Learning representations by back-propagating errors».

[10] A. Sherstinsky, «Fundamentals of Recurrent Neural Network (RNN) and Long Shor-Term Memory (LSTM) Network,» 2020.

[11] W. Zaremba, «Recurrent Neural Network Regularization,» 2015.

[12] R. Caruana, S. Lawrence y L. Giles, «Overfitting in Neural Nets: Backpropagation, Conjugate Gradient and Early Stopping,» 2001.

[13] S. Hochreiter y J. Schmidhuber, «Long short-term memory,» 1997.

[14] C. Kyunghyun, B. van Merrienboer, C. Gulcehre, D. Bhadanau, F. Bougares, H. Schwenk y Y. Bengio, «Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation,» 2014.

[15] M. Ravanelli, P. Brakel, M. Omologo y Y. Bengio, «Light Gated Recurrent Units for Speech Recognition,» 2018.

[16] G. Weiss, Y. Goldberg y E. Yahav, «On the Practical Computational Power of Finite Precision RNNs for Language Recognition,» 2018.

[17] D. Britz, A. Goldie y Q. Le, «Massive Exploration of Neural Machine Translation Architectures,» 2017.

[18] R. Puduppully, M. Lapata y L. Dong, «Data-to-Text Generation with Content Selection and Planning,» 2019.

[19] C. Gulcehre, M. Moczulski, M. Denil y Y. Bengio, «Noisy Activation Functions,» 2016.

[20] C. Oliva y L. Lago-Fernández, «Interpretability of Recurrent Neural Networks Trained on Regular Languages» *IWANN,* pp. 14-25, 2019.

Glosario

Adam	Adaptative Stochastic Method
AI	Artificial Intelligence
ANN	Artificial Neural Networks
Cell	RNN hidden state
CV	Clip value
GRU	Gated Recurrent Unit
LR	Learning Rate
LSTM	Long Short-Term Memory
RNN	Recurrent Neural Networks
SGD	Stochastic Gradient Descent
TANH	Hyperbolic Tangent

Anexos

A Búsqueda de hiperparámetros para SGD

A continuación, se muestran los resultados de pérdida en validación de la búsqueda en rejilla para los hiperparámetros del optimizador SGD.

		Learning Rate									
Clip value		5e-3	0.171	0.337	0.503	0.669	0.835	1.002	1.168	1.333	1.5
	1e-3	1.876	1.531	1.501	1.463	1.459	1.478	1.510	1.548	1.625	1.684
	1.4e-3	1.836	1.487	1.467	1.426	1.431	1.446	1.562	1.622	1.661	1.693
	1.8e-3	1.801	1.495	1.446	1.400	1.477	1.542	1.593	1.593	1.683	1.711
	2.3e-3	1.770	1.482	1.424	1.383	1.561	1.607	1.776	1.652	1.692	1.735
	2.7e-3	1.750	1.480	1.414	1.378	1.624	1.643	1.609	1.641	1.772	1.923
	3.2e-3	1.740	1.481	1.418	1.381	1.671	1.701	1.792	1.777	2.033	2.874
	3.6e-3	1.724	1.466	1.404	1.393	1.734	1.645	1.664	3.070	2.761	2.184
	4.1e-3	1.712	1.474	1.398	1.525	1.836	1.649	1.868	2.486	3.812	3.736
	4.5e-3	1.705	1.468	1.402	1.680	1.678	1.732	1.858	3.895	2.445	3.112
	5e-3	1.702	1.465	1.393	1.734	2.051	1.759	2.209	4.203	2.034	2.521

Se puede observar que el *clip value* funciona como un seguro *learning rates* altos, haciendo que la mejor combinación sea la señalada, con lr=0.503 y cv=2.7e-3, que utilizaré para todas las pruebas con SGD. Las conclusiones más importantes son que con *learning rates* bajos, el *clip value* tiene poca influencia; pero que con *learning rates* altos si que es necesario limitarlo. La prueba se realizó con 50 épocas, buscando favorecer también el tiempo de convergencia